IPG Politécnico da Guarda
Polytechnic of Guarda

RELATÓRIO DE PROJETO

Licenciatura em Engenharia Informática

Janilza Almeida Simão

novembro | 2018

Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

SISTEMA DE MONITORIZAÇÃO DA QUALIDADE DO AR EM ESPAÇOS INTERIORES

JANILZA ALMEIDA SIMÃO

RELATÓRIO PARA A OBTENÇÃO DO GRAU DE LICENCIADA

EM ENGENHARIA INFORMÁTICA

Novembro/2018

Agradecimentos

Ao meu docente orientador, professor Doutor Carlos Carreto, pelas instruções, pelo incentivo e disponibilidade, os meus sinceros agradecimentos.

À entidade acolhedora do meu estágio na pessoa do Engenheiro Ricardo Santos e Engenheira Telma Estrela pelo acolhimento e disponibilidade que me reservaram durante o período de estágio nos Serviços de Sistemas e Tecnologias da Informação e Comunicação na Unidade Local de Saúde da Guarda.

A todos os docentes de Engenharia Informática que tanto contribuíram para o meu sucesso neste percurso.

A minha família por toda a motivação que sempre deram e por sempre acreditarem em mim, um especial obrigado aos meus pais e ao meu irmão.

Ao meu namorado e aos meus amigos por sempre estarem presentes tanto nos bons momentos como nos menos bons.

Muito obrigada a todos.

Ficha de Identificação

Aluna: Janilza Almeida Simão, Nº 1012414

Email: 1012414@sal.ipg.pt

Curso: Licenciatura em Engenharia Informática

Estabelecimento de Ensino: Instituto Politécnico da Guarda, Escola Superior de Tecnologia e Gestão

Docente Orientador: Professor Doutor Carlos Carreto

Grau académico: Doutorado

Email: ccarreto@ipg.pt

Ano Letivo: 2017/2018

Instituição acolhedora do estágio: Unidade Local de Saúde da Guarda, Hospital Sousa Martins

Supervisores: Eng. Ricardo Santos e Eng. Telma Estrela

Resumo

Este relatório descreve o trabalho realizado no âmbito do estágio curricular no Serviço de Sistemas e Tecnologias da Informação e Comunicação - SSTIC da Unidade Local de Saúde da Guarda – ULSG. O trabalho consistiu no desenvolvimento de um sistema de monitorização da qualidade do ar interior, demonstrando os conhecimentos adquiridos ao longo da licenciatura em Engenharia Informática.

O objetivo foi de monitorizar alguns parâmetros físicos e químicos do ar nomeadamente, temperatura, humidade, sensação térmica, ponto de orvalho, dióxido de carbono, partículas em suspensão e compostos orgânicos voláteis. Com recurso a um minicomputador *LattePanda* e a um modulo de quatro sensores construiu-se uma aplicação em C# para apresentar os valores dos parâmetros do ar medidos. Estes também estão disponíveis na plataforma *ThingSpeak.com* onde são apresentados sob forma de gráfico e podem ser consultados em qualquer navegador por se tratar de um serviço na *cloud.* O sistema também possui o histórico das medições armazenados numa base de dados MySQL.

Palavras chave: Qualidade do Ar Interior**,** C#, *LattePanda, MySQL, ThingSpeak.*

Abstract

This report describes the work developed within the scope of the internship at Information and Communication Systems and Technologies Service on Local Health Unit of *Guarda* - ULSG, which consisted on development of an indoor air quality monitoring system, demonstrating the knowledge acquired during the degree in Computer Engineering.

The objective was to monitor some physical and chemical parameters of air such as temperature, humidity, heat index, dew point, carbon dioxide, particulate matter and volatile organic compounds. Using a LattePanda computer and a four-sensor module, a C # application was building to present the values of the measured air parameters. These are also available on the *ThingSpeak.com* platform presented in graph form and can be consulted over any browser because it is a cloud-based service. System also has the history of measurements stored on MySQL database.

Keywords: Indoor Air Quality, C#, *Lattepanda*, *MySQL*, *ThingSpeak.*

Índice Geral

Agradecimentos iii
Ficha de Identificação v
Resumo vii
Abstract ix
Índice Geral xi
Índice de Figuras xiii
Índice de Tabelas xiv
Lista de siglas e acrónimos xv
1. Introdução 1
1.1. Caracterização sumária da instituição 1
1.1.1. Serviço de Sistemas e Tecnologias da Informação e Comunicação da ULSG - SSTIC 2
1.2. Motivação 3
1.3. Descrição do problema 4
1.1. Solução e Objetivos 5
1.2. Plano de Estágio 6
1.3. Metodologia 7
1.4. Estrutura do documento 9
2. Estado da Arte 11
2.1. Qualidade do ar interior 11
2.1.1 Problemas associados a má qualidade do ar interior 11
2.1.2. Componentes e respetivos valores de referência 13
2.2. Soluções existentes 14
2.2.1 Projetos de Investigação e Desenvolvimento 14
2.2.2. Produtos comerciais 19
3. Definição de requisitos 23
3.1. Requisitos funcionais 23
3.2. Requisitos não funcionais 24
4. Implementação 25
4.1. Tecnologias 25
4.2. Arquitetura do hardware 26
4.2.1. Sensor de Partículas em Suspensão - $\boldsymbol{PM2.5}$ 27

4.2.2. Sensor de Dióxido de carbono - $\boldsymbol{CO2}$ ... 28
4.2.3. Sensor de Compostos Orgânicos Volateis - $\boldsymbol{VOC}$ ... 28
4.2.4. Sensor de temperatura e humidade relativa ... 28
4.2.5. Sensor virtual para obter o ponto de orvalho ... 28
4.2.6. Sensor virtual para obter a sensação térmica ... 29
4.3. Arquitetura do *software* ... 31
4.3.1. Módulo AirQuality.ino ... 32
4.3.2. Modulo *FrmPrincipal* ... 36
4.3.3. Modulo *ArduinoCOM* ... 36
4.3.4. Modulo *SensorValue* ... 37
4.3.5. Modulo *AppAirQuality* ... 37
4.3.6. Modulo *MySqlDBConnection* ... 39
4.3.7. Modulo *SendEmail* ... 42
4.3.8. Modulo *ThingSpeak* ... 43
4.4. Protótipo ... 45
5. Verificação e Validação ... 47
6. Conclusões ... 50
6.1. Trabalho futuro ... 51
Referências ... 52
Anexos ... 54
A 6.1 Arquitetura do *hardware* do *lattepanda* ... 56
A 6.2 ArduinoCOM.cs ... 59
A 6.3 SensorValue.cs ... 61
A 6.4 AppAirQuality.cs ... 65
A 6.5 MySQLDBConnection.cs ... 76
A 6.6 SendEmail.cs ... 79
A 6.7 ThingSpeak.cs ... 82

Índice de Figuras

Figura 1.1 – Estrutura organizacional do SSTIC 2
Figura 1.2 – Representação gráfica do plano de estágio 6
Figura 2.1 - Arquitetura do sistema de monitorização e controlo da Qualidade do Ar 15
Figura 2.2 - Protótipo do monitor e controlador da qualidade do ar interior 16
Figura 2.3- - Arquitetura do iAQ 16
Figura 2.4 - Protótipo do iAQ Fonte 17
Figura 2.5 - Arquitetura do sistema de monitorização e controlada qualidade do ar 18
Figura 2.6 – a) Sensor Unit Hardware, b) Control Unit Hardware 18
Figura 2.7 - Air Fluke AirMeter 19
Figura 2.8 – a) Exemplo de ecrãs, b) Aparelho AdvancedSense Pro 20
Figura 2.9 - Airthinx aparelho 21
Figura 2.10 - Plataformas para aceder ao AirThinx 21
Figura 4.1 – Arquitetura do hardware 26
Figura 4.2 - Módulo de Sensores 27
Figura 4.3 – Arquitetura do Software 31
Figura 4.4 – Diagrama de classes da aplicação C# 35
Figura 4.5 - Ecrã da aplicação 39
Figura 4.6 – Gráficos gerados pelo ThingSpeak 44
Figura 4.7 – Protótipo obtido 45
Figura 5.1 – Teste do sensor de compostos orgânicos voláteis (VOCs) 47
Figura 5.2 – Email de alerta VOC 47
Figura 5.3 – Teste do sensor de Partículas em suspensão (PM2.5) 48
Figura 5.4 – Alerta PM2.5 48
Figura 5.5 – Teste do Sensor de Humidade Relativa Antes 49
Figura 5.6 - Teste do Sensor de Humidade Relativa Depois 49

Índice de Tabelas

Tabela 1.1 – Plano de Estágio 6
Tabela 2.1 - Características dos poluentes e efeitos na saúde 11
Tabela 2.2 - Limiar de proteção e margem de tolerância para os poluentes físico-químicos 13
Tabela 3.1 – Requisitos funcionais 23
Tabela 3.2 - Requisitos não funcionais 24
Tabela 4.1- Tabela com as características do módulo de sensores 30

Lista de siglas e acrónimos

AVAC	Aquecimento Ventilação e Ar Condicionado
CO_2	Dióxido de Carbono
CO	Monóxido de carbono
COV	Compostos Orgânicos não Voláteis
CH_2O	Formaldeído
IACS	Infeções Associadas aos Cuidados de Saúde
IDE	*Integrated Development Environment*
NDIR	*Non-Dispersive Infrared*
OMS	Organização Mundial da Saúde
$PM_{2.5}$	Partículas em Suspensão com diâmetro inferior a 2,5µm
PM_{10}	Partículas em Suspensão com diâmetro inferior a 10µm
PHP	*Hypertext Preprocessor*
SED	Síndrome do Edifício Doente
SSTIC	Serviço de Sistemas e Tecnologias da Informação e Comunicação
SQL	*Structured Query Language*
SMTP	*Simple Mail Transfer Protocol*
UDI	Unidade de investigação para o Desenvolvimento do Interior
ULSG	Unidade Local de Saúde da Guarda
UTA	Unidade de Tratamento do Ar
VNC	*Virtual Network Computing*

1. Introdução

Neste primeiro capítulo é feita uma apresentação da instituição onde foi realizado o estágio e desenvolvido o projeto descrito neste relatório. São também apresentados os motivos que favoreceram a origem do projeto, a descrição do problema e os objetivos que se pretende alcançar, o plano do estágio que inclui tanto as etapas como a duração prevista das mesmas e por fim uma breve descrição da estrutura do documento.

1.1. Caracterização sumária da instituição

O Hospital Sousa Martins surgiu na década de 70 e desde então tem como atividade principal a prestação de cuidados de saúde primários, diferenciados e continuados à população.

Inicialmente tratava-se de um sanatório para tratamento da tuberculose e foi atribuído o nome de Hospital Sousa Martins em homenagem a um médico, com o mesmo nome, pela sua dedicação na cura da mesma.

Nas últimas décadas o hospital Sousa Martins funcionou como hospital distrital com múltiplas especialidades e em 2008 foi constituída a Unidade Local de Saúde da Guarda - ULSG onde estão incluídas todas as instituições de saúde do distrito exceto Aguiar da Beira.

A Missão da ULSG, traduz-se na prestação de cuidados de saúde à comunidade, numa ótica de melhoria contínua, através da prossecução de padrões de excelência nos cuidados aos utentes, nomeadamente através da [1]:

- Prestação da melhor qualidade de cuidados e serviços à comunidade, na prevenção, diagnóstico e tratamento das patologias humanas;
- Cooperação e participação com os estabelecimentos de ensino superior, a nível regional, nacional e internacional, no apoio e fomento da educação dos profissionais de saúde, bem como, da investigação e pesquisa nas áreas clínicas;
- Atração e manutenção de profissionais motivados e com elevadas competências técnicas;

- Participação ativa na comunidade envolvente, com vista ao incremento dos níveis de saúde e bem-estar, dos atuais e potenciais utentes.

O estágio foi realizado nos Serviços de Sistemas e Tecnologias da Informação e Comunicação – SSTIC da ULSG e consiste no desenvolvimento de um sistema informático para monitorizar a qualidade do ar de um espaço interior e apresentar os resultados desse monitoramento de duas formas distintas (pública e não pública).

1.1.1. Serviço de Sistemas e Tecnologias da Informação e Comunicação da ULSG - SSTIC

O objetivo fundamental do SSTIC é o de assegurar o bom funcionamento da infraestrutura de rede informática da ULSG, no âmbito dos meios computacionais e serviços de comunicações existentes, do seu parque aplicacional e tecnológico, bem como o fornecimento de ferramentas de apoio às tomadas de decisão, fiáveis e em tempo útil, necessárias à prossecução das atividades da ULS da qual faz parte [2]. Na *Figura 1.1* é possível ver a estrutura organizacional dos SSTIC.

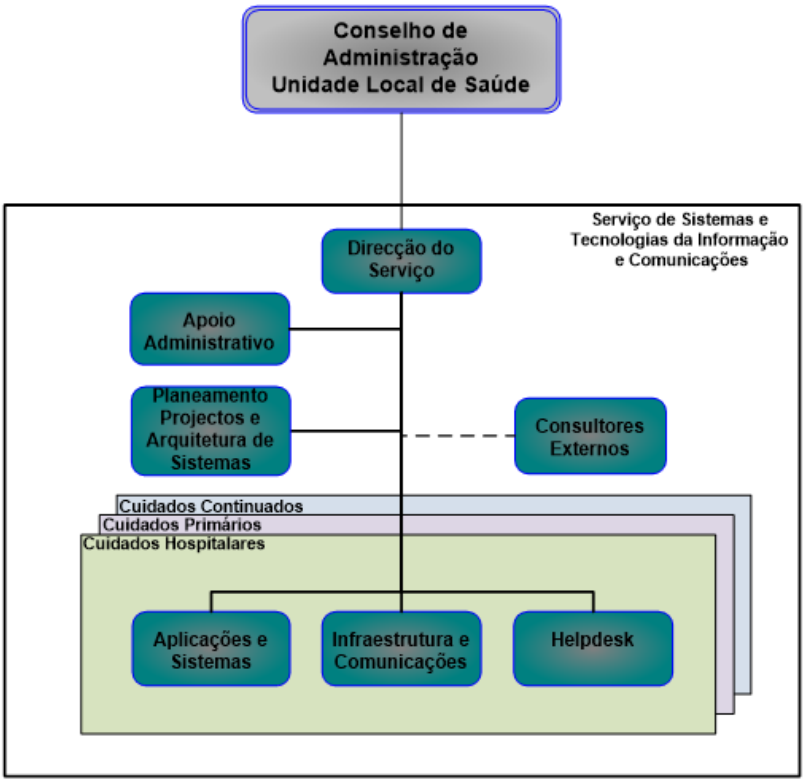


Figura 1.1 – Estrutura organizacional do SSTIC

1.2. Motivação

Atualmente, é cada vez mais comum a preocupação com a qualidade do ar que se respira no interior dos edifícios por se tratar de um dos fatores determinantes para uma vida saudável, uma vez que as pessoas passam grande parte do tempo no interior dos mesmos [3]. Na maior parte das situações a qualidade do ar interior é pior do que a do ar exterior.

Ao contrário do que muitas vezes se pensa, o problema da má qualidade do ar interior surgiu desde tempos remotos quando o Homem começou a usar o fogo nos espaços fechados. Hoje, associado a grandes concentrações de pessoas no mesmo espaço e outros fatores físicos, químicos, microbiológicos e radioativos, a qualidade do ar interior é um tema muito sensível.

Devido a diversos fatores como: grande concentração de pessoas no mesmo espaço, más condições dos equipamentos de Aquecimento, Ventilação e Ar Condicionado - AVAC; má qualidade do ar exterior; libertação de diferentes tipos de gases provenientes das tintas das paredes, isolamento, adesivos, revestimentos de piso; a qualidade do ar no interior dos edifícios tem-se degradado cada vez mais e quando se refere aos hospitais ou outras instituições ligadas a saúde, a situação é mais alarmante pois tratam-se de pessoas com baixa imunidade.

Apesar de já existirem sistemas para a monitorização da qualidade do ar e de haver empresas especializadas que efetuam inspeção nos hospitais e outras repartições públicas, ainda existem poucos edifícios com uma monitorização continua dos parâmetros que afetam a qualidade do ar interior. Esse número deve-se ao facto de se tratar de um tema que ainda é pouco relevante aos olhos de muitos e pelos custos elevados associados.

Pelo facto de um centro hospitalar se tratar de um local muito frequentado diariamente, é necessário ter uma atenção especial à qualidade do ar do seu interior. Uma má qualidade do ar, põe em causa não só a saúde dos pacientes e dos profissionais de saúde, mas também a dos visitantes e outras pessoas que frequentam o local ocasionalmente. Conviver diariamente num ar poluído pode causar diversos sintomas, mesmo às pessoas saudáveis, como: dores de cabeça, fadiga, falta de ar, congestão nasal, tosse, espirros, irritação nos olhos, nariz, garganta; tonturas, náuseas e muitos outros sintomas. Quando se trata de pacientes, os sintomas podem ser mais graves. Por exemplo se se tratar de um paciente com problemas respiratórios ou com a imunidade muito frágil, inspirar um ar poluído pode mesmo levar a óbito.

Assim sendo, surgiu a ideia de desenvolver um sistema de monitorização da qualidade do ar, a baixo custo, composto por sensores que recolham informação relativa a determinados gases e partículas que se encontram no ar de um espaço interior do hospital para que esses dados sejam posteriormente analisados e se possa adotar medidas corretivas adequadas.

1.3. Descrição do problema

Como já foi referido no ponto anterior, não existe uma monitorização contínua da qualidade do ar interior no Hospital Sousa Martins. Apesar de existirem inspeções esporádicas, estas não são suficientes uma vez que o objetivo principal do projeto é melhorar a qualidade do ar do hospital, com os dados atuais não se consegue ter uma perceção real do tipo de ar que se encontra no seu interior diariamente.

É necessária uma monitorização contínua de forma que, em conjunto com as Unidades de Tratamento do Ar - UTA, seja possível fornecer um ar de qualidade em conformidade com os padrões exigidos. Para tal, pretende-se desenvolver um sistema que faça a monitorização contínua da qualidade do ar interior em determinadas áreas do hospital.

Numa primeira fase será implementado em áreas menos críticas como por exemplo as salas de espera e futuramente espera-se que se possa estender para áreas mais críticas como o centro cirúrgico, salas de recobro e os restantes setores inseridos na unidade hospitalar. Por se tratar de um hospital, será imprescindível fazer um estudo em relação ao tema de forma a selecionar os componentes que se pretende analisar bem como os sensores que vão ser utilizados para efetuar as medições.

1.1. Solução e Objetivos

A solução proposta para o problema descrito na secção anterior, foi o desenvolvimento de um sistema informático formado por um minicomputador, um modulo de sensores e um monitor LCD, capaz de monitorizar a qualidade do ar de um espaço interior e apresentar os resultados desse monitoramento de forma pública (no monitor LCD) e não pública, enviando os dados para uma base de dados local e para um serviço na *cloud*, de modo a ser criado um histórico dos dados para análise posterior.

Foram definidos os seguintes objetivos correspondentes às características e funcionalidades finais do sistema:

- Desenvolver um protótipo funcional do sistema de baixo custo e de fácil instalação e manutenção;
- Desenvolver uma interface gráfica para o sistema, capaz de apresentar os dados referentes à qualidade do ar, de forma apelativa e informativa para o público em geral;
- Desenvolver um *BackOffice* para armazenamento do histórico das medições bem como a gestão de alarmes.

1.2. Plano de Estágio

A *Tabela 1.1* apresenta o plano de estágio e inclui a descrição das etapas bem como as datas previstas para início e término e as respetivas durações.

Tabela 1.1 – Plano de Estágio

Etapa	Designação	Início	Término
1	Definição do problema e estudo do estado da arte	28/05/2018	15/06/2018
2	Especificação de requisitos e proposta de solução	18/06/2018	29/06/2018
3	Desenvolvimento da solução e construção de um protótipo	02/07/2018	31/08/2018
4	Teste do protótipo	02/09/2018	14/09/2018
5	Elaboração do relatório de estágio	28/05/2018	21/09/2018

De modo a ter uma melhor perceção da duração das etapas, apresenta-se um gráfico na *Figura 1.2*

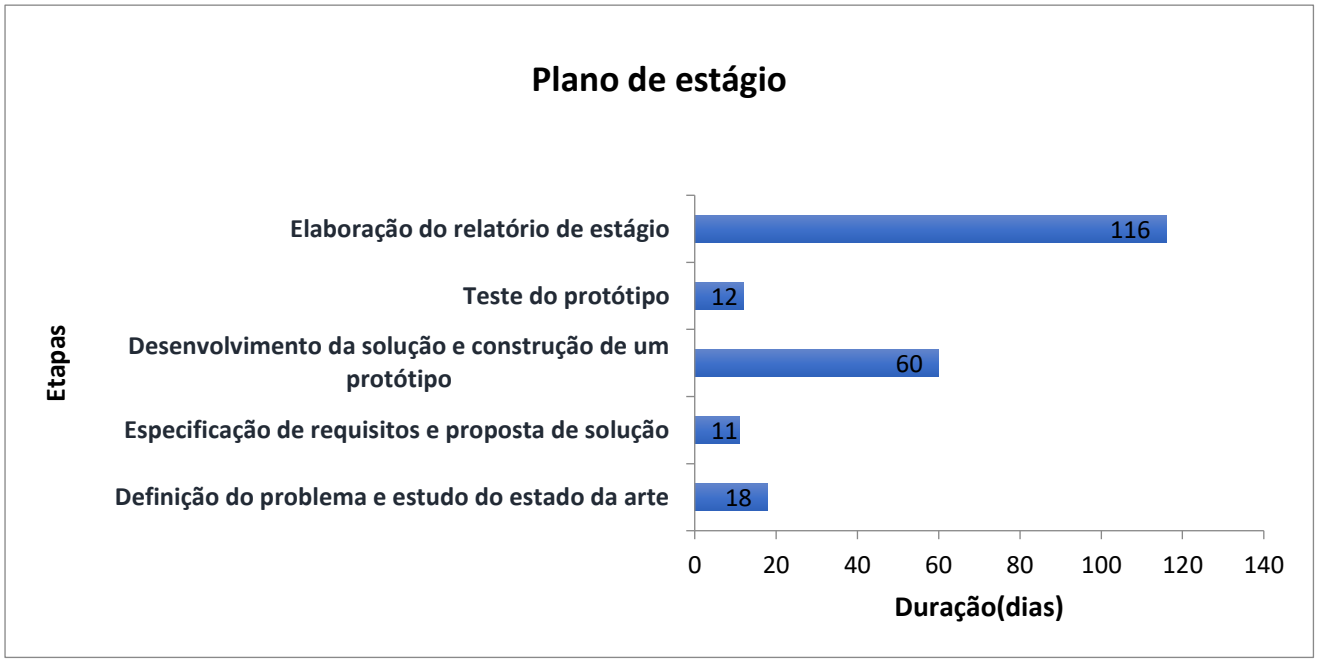


Figura 1.2 – Representação gráfica do plano de estágio

1.3. Metodologia

A metodologia adotada para o projeto é a metodologia geral de desenvolvimento de um projeto de informática que envolve:

1. **Definição do problema**

   Nesta etapa é feita a formulação do problema e só está correta quando responde as seguintes questões:

   - Qual é o problema?
   - Qual é a origem do problema?
   - Quem tem o problema?

2. **Estudo do estado da arte**

   Depois de dadas as respostas da etapa anterior é preciso começar a procurar soluções. Assim sendo efetua-se pesquisas em relação ao tema de forma a adquirir mais conhecimentos a respeito do mesmo e de encontrar as soluções já existentes. Pode-se dividir essas soluções em projetos de investigação e desenvolvimento que geralmente são documentados em artigos científicos e projetos comerciais.

3. **Especificação de requisitos**

   Nesta fase são definidas as funcionalidades que se pretende que o produto tenha. Essa escolha é geralmente feita com base nas necessidades do cliente/utilizador e no estudo efetuado anteriormente no estado da arte.

4. **Definição da solução**

   Nesta fase é feita a escolha da melhor forma de implementar os requisitos do sistema. São realizadas reuniões entre as partes intervenientes com o objetivo de decidir qual é a melhor solução e as tecnologias a ser utilizadas no seu desenvolvimento

5. **Desenvolvimento da solução e construção de um protótipo**

   É a fase de implementação da solução onde também é feito o desenvolvimento do protótipo de forma a posteriormente se proceder ao teste da mesma.

6. **Teste do protótipo**

   São feitos vários tipos de teste ao sistema e em função dos resultados obtidos podem ser necessárias alterações aos requisitos do sistema, definição de uma nova solução fazer alterações no código e voltar a testar, até que se obtenha a um protótipo que cumpra os requisitos definidos.

7. **Elaboração do relatório do projeto**

   A elaboração do relatório do projeto deve acompanhar todo o processo de desenvolvimento da solução.

1.4. Estrutura do documento

No capítulo 1 - Introdução apresenta-se uma breve descrição da instituição de saúde, da motivação para desenvolver o projeto, descrição do problema, objetivos, o plano de estágio previsto que inclui a descrição das tarefas a executar e as respetivas durações bem como a metodologia a ser utilizada no desenvolvimento do projeto. No capítulo 2, Estado da Arte, é descrita a pesquisa efetuada sobre a qualidade do ar interior e apresentação de soluções já existentes no mercado. De seguida define-se os requisitos funcionais e não funcionais do sistema presentes no capítulo 3 e no 4 apresenta-se uma implementação da solução definida ao longo dos capítulos anteriores e as tecnologias utilizadas. No penúltimo capítulo é feita a verificação e validação do sistema implementado e por fim são tiradas as conclusões no capítulo 6.

No final do documento, é possível consultar as referências bibliográficas e respetivos anexos citados ao longo do mesmo

2. Estado da Arte

Neste capítulo é descrita a pesquisa realizada acerca da qualidade do ar interior, que incluirá os problemas associados a uma má qualidade do ar interior, os elementos físicos e químicos que devem ser monitorizados, bem como a análise de alguns projetos de investigação e desenvolvimento e projetos comerciais semelhantes ao que foi desenvolvido.

2.1. Qualidade do ar interior

Segundo Organização Mundial da Saúde – OMS, a qualidade do ar interior constitui um dos principais riscos para a saúde pública [4].Dada a crescente preocupação em relação ao tema, a OMS publicou, em 2010, um guia europeu que contém a avaliação dos riscos para a saúde de vinte e oito contaminantes químicos do ar "*WHO guidelines for indoor air quality: selected pollutants"* [5].

2.1.1 Problemas associados a má qualidade do ar interior

Na tabela que se segue apresentam-se alguns dos poluentes que mais afetam a qualidade do ar, as suas características e os efeitos que causam na saúde [6].

Tabela 2.1 - Características dos poluentes e efeitos na saúde

Poluentes	Características físico-químicas	Efeitos na saúde
Partículas em suspensão (fração $\boldsymbol{PM_{2.5}}$ $\boldsymbol{e}$ $\boldsymbol{PM_{10}}$)	Material sólido ou pequenas gotículas de vapor, fumo e poeiras	• Dores de cabeça • náuseas, cansaço • Carboxihemoglobinemia (impede a captação de oxigénio) • Efeitos no sistema nervoso central e no sistema cardiovascular
Compostos Orgânicos Voláteis (VOCs)	Solventes de uso comum (benzeno, tolueno, xileno, tricloroetileno, tetracloroetileno, odores entre outros)	• sintomas de alergia • náuseas • vertigens • fadiga • dores de cabeça • olhos vermelhos • secura das mucosas do nariz e garganta • cancro da pele e do pulmão, leucemia

Monóxido de carbono $\boldsymbol{CO}$	Incolor; inodoro	• Dores de cabeça náuseas, cansaço • Carboxihemoglobinemia (impede a captação de oxigénio) • Efeitos no sistema nervoso central e no sistema cardiovascular
Formaldeído($\boldsymbol{CH_2O}$)	Incolor, odor forte, solúvel em água, muito reativo	• Irritação dos olhos, nariz, garganta • Dificuldades respiratórias, enjoos • Fadiga
Dióxido de Carbono($\boldsymbol{CO_2}$)	Incolor	• Dores de cabeça • Irritação dos olhos e garganta, fadiga, falta de ar • Efeitos no sistema nervoso central e no sistema cardiovascular
Radão	Incolor, inodoro e insípido;	• Cancro do pulmão • Leucemia Infantil

O termo *Sick Building Syndrome – SBS* definido pela primeira vez pela OMS, na qualidade de entidade de coordenação, monitorização e avaliação da saúde pública, é usado para descrever uma condição médica em que as pessoas apresentam sintomas variados causados pela permanência no interior de determinados edifícios. “Talvez o doente não seja a pessoa, mas sim o local onde se encontra” [6].

Geralmente manifesta-se através de diversos sintomas como: comichão, cansaço, vermelhidão, nariz irritado, secura ou dor de garganta e muitos desses sintomas são causados por alguns sinais pela má qualidade do ar como: ar fraco, ar seco, luz ou temperaturas muito quentes ou muito frias [7].

Igualmente definido pela OMS, o termo *Health Care-associtated Infection – HCAI* também referido como infeção "hospitalar" ou "nosocomial”, é uma infeção ocorrida num doente durante o processo de cuidados num hospital ou noutra instalação de cuidados de saúde que não estava presente ou incubada durante a sua triagem. Normalmente são detetadas nos pacientes após a alta, mas também podem ocorrer nos profissionais de saúde e nesse caso são referidas como infeções ocupacionais.

De entre outros fatores que originam a *HCAI,* um deles é a má qualidade do ar [8].

2.1.2. Componentes e respetivos valores de referência

Para efetuar uma boa monitorização da qualidade do ar interior é necessário medir alguns componentes físicos e químicos. Na *Tabela 2.2* que se segue apresenta-se tanto alguns componentes como os respetivos valores de referência [9].

Tabela 2.2 - Limiar de proteção e margem de tolerância para os poluentes físico-químicos

Poluentes	**Unidade**[1]	**Limiar de Proteção**[2]	**Margem de Tolerância MT (%)**[3]
Partículas em suspensão (fração $PM_{2.5}$)	[µg/m3] [4]	50	100
Partículas em suspensão (fração PM_{10})	[µg/m3]	25	100
Compostos Orgânicos Voláteis (COV)	[µg/m3]	600	100
Monóxido de carbono (CO)	[mg/m3] [ppmv][5]	10 9	-
Formaldeído (CH_2O)	[µg/m3] [ppmv]	100 0.08	-
Dióxido de Carbono (CO_2)	[mg/m3] [ppmv]	2250 1250	30
Radão	[Bq/m3][6]	400	-

[1] As concentrações referem-se à temperatura de 20 °C e à pressão de 1 atm (101,325 KPa)
[2] Os limiares de proteção indicados dizem respeito a uma média de 8 horas
[3] As margens de tolerância são aplicadas a edifícios existentes e edifícios novos sem sistemas mecânicos de ventilação
[4] Microgramas por metro cúbico
[5] Partes por milhão por volume
[6] Beckerel por metro cúbico

A temperatura e a humidade também constituem fatores físicos que podem afetar de forma negativa a qualidade do ar, quando excedem os parâmetros de conforto térmico geralmente aceites que variam entre 20º-26º para a temperatura e 40%-65% para a humidade relativa [10].

2.2. Soluções existentes

De modo a adquirir mais conhecimentos acerca do tema e da implementação do sistema pretendido, foi feita uma pesquisa em relação a projetos de investigação e desenvolvimento e produtos comerciais já desenvolvidos.

A seguir apresenta-se algumas das soluções analisadas.

2.2.1 Projetos de Investigação e Desenvolvimento

O projeto que se segue consiste na construção, validação e aplicação de um sistema de monitorização da qualidade do ar a baixo custo e de fácil instalação.

Como é possível ver na *Figura 2.1* [3], a arquitetura do sistema é composta por 3 módulos:

1. Módulo sensorial (**A**) para medir parâmetros químicos (Dióxido de carbono - $\mathbf{CO_2}$ e Monóxido de Carbono - $\boldsymbol{CO}$,) e físicos (temperatura e humidade relativa);

2. Unidade de aquisição e de controlo constituída por um microcontrolador *Arduino* (**B**) e um módulo *Wi-Fi* (**E**) que permite o envio e a consulta dos dados em tempo real;

3. Um atuador (**C**) que ativa o ventilador (**D**) em função das concentrações de $\mathbf{CO_2}$ e $\boldsymbol{CO}$ (**D)**.

Toda a informação colhida pelos sensores é armazenada numa base de dados não relacional da plataforma *Carriots*[7].

O sistema possui ainda uma página web desenvolvida nas linguagens HTML, CSS e *javascript* que permite que os utilizadores acedam aos dados para consulta e recebam alertas. Estes alertas também são enviados por mensagens para o telemóvel.

[7] Trata-se de uma plataforma desenvolvida para o alojamento e desenvolvimento de aplicações ligadas a *internet* das coisas.

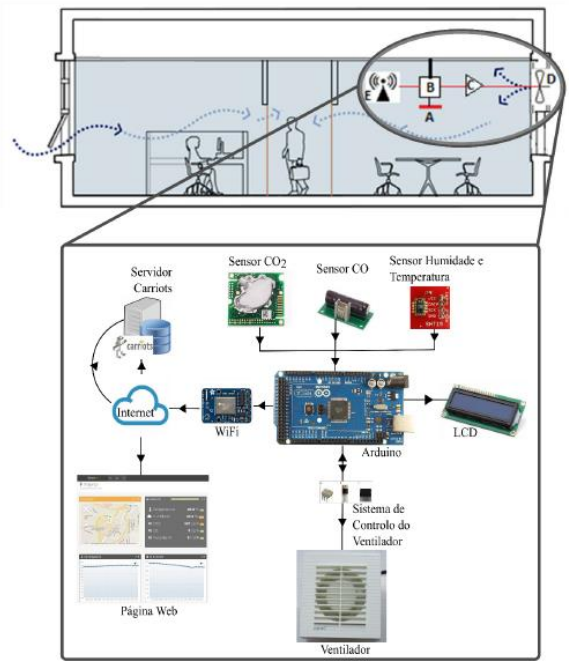


Figura 2.1 - Arquitetura do sistema de monitorização e controlo da Qualidade do Ar

Foram realizados 4 ensaios, que consistiram na medição dos parâmetros em algumas salas sob as seguintes condições:

1. Sem ventilação;
2. Com ventilação natural;
3. Com ventilação mecânica simples que consistiu na insuflação de ar para o interior da sala de acordo com os níveis de $\mathbf{CO_2}$, e $\boldsymbol{CO}$, medidos;
4. Com ventilação mecânica simples que consistiu na exaustão de ar para o interior da sala de acordo com os níveis de $\mathbf{CO_2}$, e $\boldsymbol{CO}$, medidos;

Na *Figura 2.2* apresenta-se o protótipo do sistema desenvolvido.

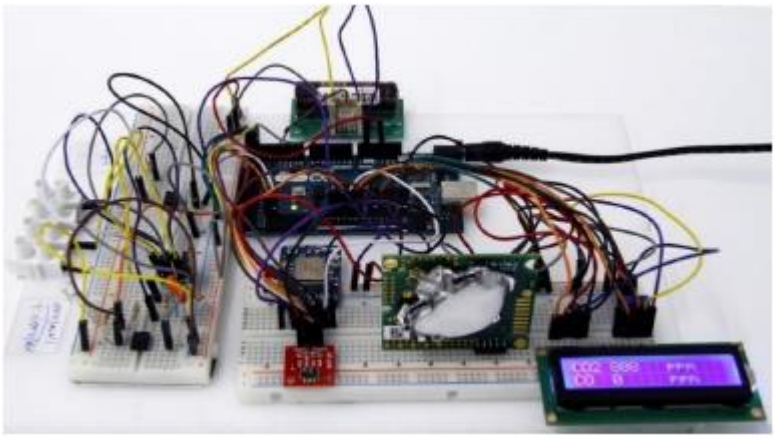


Figura 2.2 - Protótipo do monitor e controlador da qualidade do ar interior

O iAQ [18] é um sistema desenvolvido na UDI – Unidade de Investigação para o Desenvolvimento do Interior, Instituto Politécnico da Guarda de dimensões reduzidas ($20 * 10\ cm^2$), baixo custo de construção, facilidade de instalação e acesso através de uma aplicação móvel.

O sistema é composto por 4 módulos apresentados na *Figura 2.3*:

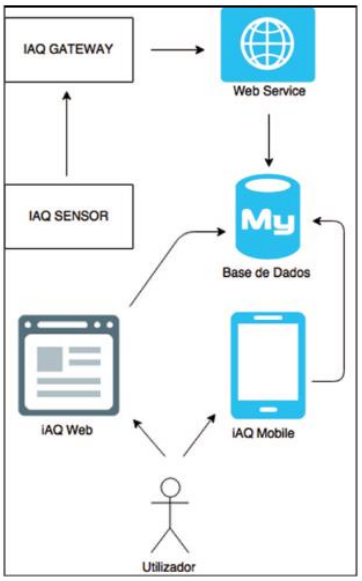


Figura 2.3- - Arquitetura do iAQ

1. O módulo *iAQ Sensor* responsável pela recolha e transferência de informação, é constituído por uma unidade de processamento, um conjunto de sensores (temperatura,

humidade, $\mathbf{CO_2}$, $\boldsymbol{CO}$, luminosidade) e um módulo para comunicação sem fios que usa o protocolo *ZigBee*[8]*;*

2. O módulo *iAQ Gateway*, nó central do controlador, está ligado a um servidor *web* de acesso remoto que permite o armazenamento e acesso aos dados (Base de dados *MYSQL*) em tempo real;

3. O módulo *iAQ Mobile* corresponde a uma aplicação que permite a visualização dos dados do sistema, receber notificações quando um determinado parâmetro ultrapassa os valores mínimo e máximo pré-definidos. Esta foi desenvolvida no *Android Studio* com recurso a linguagem de programação Java e possui mecanismos de autenticação e de proteção dos dados;

4. O módulo *iAQ Web* corresponde a uma página *Web* programada em *Hypertext Preprocessor* - PHP que permite a visualização dos dados e do respetivo histórico.

O sistema possui a capacidade de expansão por adição de mais sensores e na *Figura 2.4* que se segue apresenta-se o protótipo do sistema.

Figura 2.4 - Protótipo do iAQ Fonte

O projeto que se segue apresenta um sistema que visa monitorizar e controlar a qualidade do ar interior de edifícios inteligentes (*Smart Houses*) [11].

É composto por 4 módulos distintos (ver *Figura 2.5*):

[8] O protocolo *ZigBee* é um protocolo de comunicação sem fios de baixo consumo, baixo custos de instalação e que suporta várias configurações de rede muito utilizado na automação de edifícios [19]

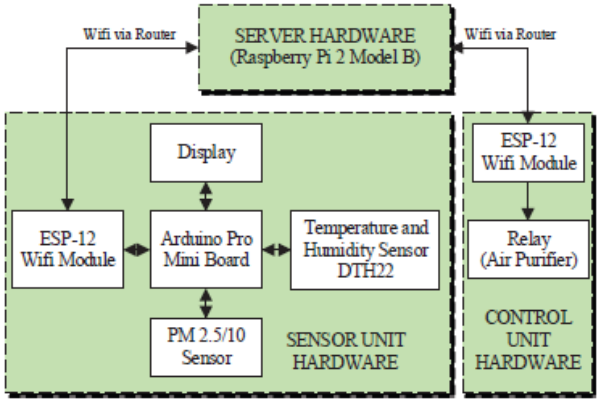


Figura 2.5 - Arquitetura do sistema de monitorização e controlada qualidade do ar

1. *Server unit hardware* constituído por um *Raspberry Pi 2 Model B* com o sistema operativo *Arch Linux ARM.* Utiliza o protocolo de transporte e um servidor central de nome *broker* para fazer a gestão das conexões entre os diferentes clientes*;*

2. *Sensor unit hardware* composto por sensores $\boldsymbol{PM_{2.5}}$ e $\boldsymbol{PM_{10}}$, humidade, temperatura,

3. um *arduino*, um ecrã LCD de 2.8" e um módulo *Wi-Fi*;

4. *Control unit hardware* com um purificador de ar e um módulo *Wi-Fi* [11].

Na *Figura 2.6* apresenta-se o protótipo do *sensor unit hardware* e *control unit hardware*.

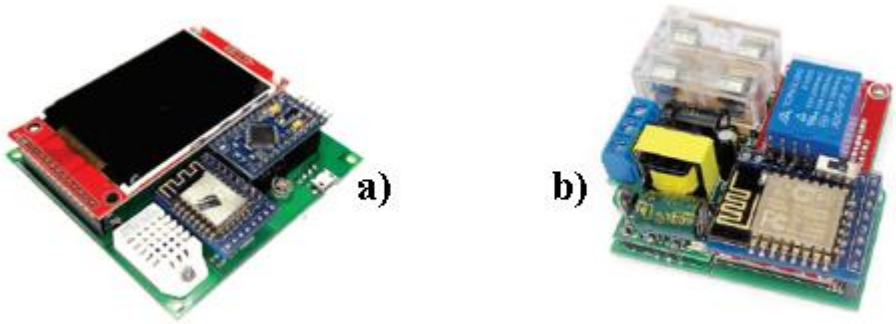


Figura 2.6 – a) Sensor Unit Hardware, b) Control Unit Hardware

2.2.2. Produtos comerciais

Atualmente já existem alguns produtos que permitem uma monitorização contínua e ininterrupta da qualidade do ar, mas é mais comum encontrar aparelhos individuais para a medição de cada componente. Após uma pesquisa abrangente, foram selecionados três produtos existentes no mercado, em que apenas um deles corresponde ao que se espera atingir no final deste projeto.

O ***Fluke 975 AirMeter*** um aparelho portátil que mede, armazena e mostra os valores de $\mathbf{CO_2}$, $\boldsymbol{CO}$, temperatura e humidade relativa. Possui alarmes visuais e audíveis e é possível transferir os valores para o PC através de uma porta USB de forma a poder obter um histórico de medições e visualização gráfica das mesmas [12].

Na *Figura 2.7* é possível ver o *Fluke 975 Air Meter.*

Figura 2.7 - Air Fluke AirMeter

*O **Advanced Sense Pro*** trata-se de um produto fabricante *Wolfsense* que permite efetuar medições de alguns parâmetros como: $\mathbf{VOC}$, $\mathbf{CO_2}$, temperatura, humidade relativa e concentração de partículas em suspensão.

Na *Figura 2.8* apresenta-se o *Advanced Sense Pro* e alguns ecrãs com medições, tratamento dos dados sob forma de gráfico. O aparelho também está equipado com um módulo *Wi-Fi* de modo a permitir o acesso remoto aos dados.

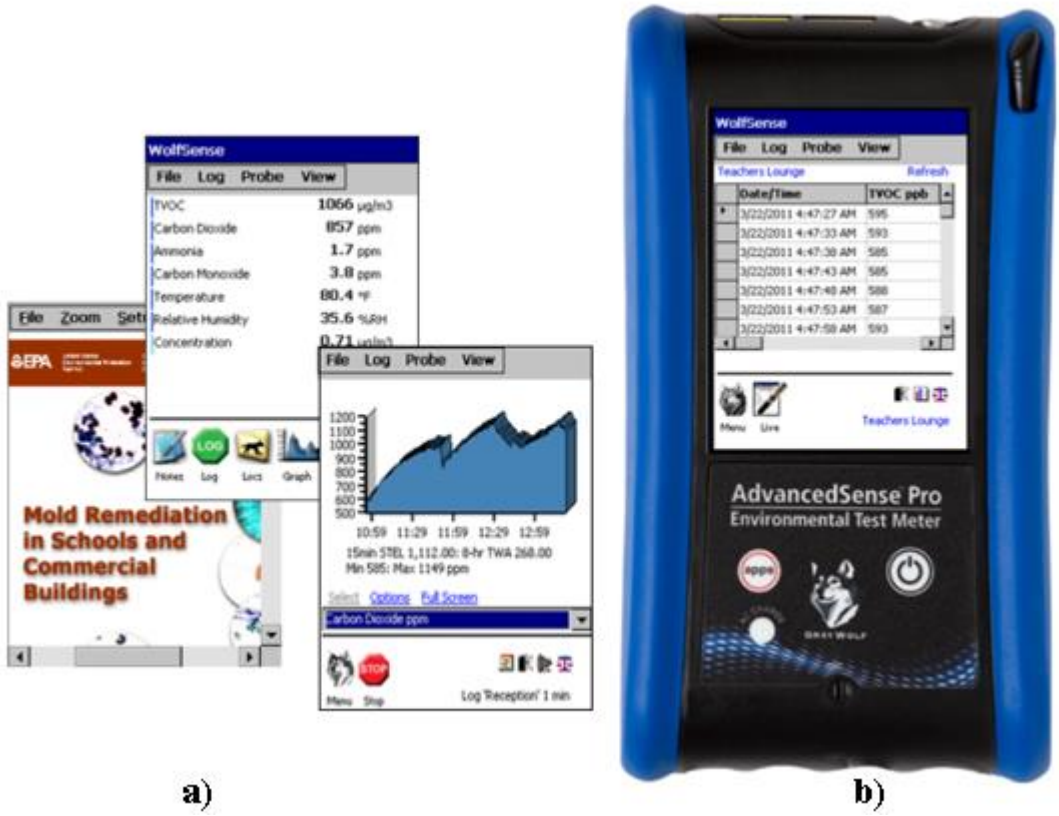


Figura 2.8 – a) Exemplo de ecrãs, b) Aparelho AdvancedSense Pro

O ***AirThinx*** é uma solução de monitorização contínua da qualidade do ar desenvolvida pela *Neutronix*. Trata-se de um sistema sem fios *cloud-based* com nove sensores integrados ($\mathbf{PM_1}, \mathbf{PM_{2.5}}, \mathbf{PM_{10}}, \mathbf{CO_2}, \mathbf{CO}, \mathbf{CH_2O}, \mathbf{VOCs}, \mathbf{Temperatura}$, **Humidade Pressão do ar**) e fornece aos utilizadores medições precisas e alertas em tempo real de modo a tornar os ambientes internos mais saudáveis [13].

Na figura abaixo apresenta-se o aparelho que efetua as medições.

Figura 2.9 - Airthinx aparelho

Os valores medidos estão disponíveis na *cloud* e podem ser acedidos através de diversas plataformas como é possível ver na *Figura 2.10.*

Figura 2.10 - Plataformas para aceder ao AirThinx

De todas as soluções analisadas, tanto os projetos de investigação e desenvolvimento comerciais, o *AirThinx* é a mais completa pois efetua medições de um maior número de parâmetros do ar e possui mais funcionalidades.

Na solução construída, optou-se por um sistema de monitorização de dióxido de carbono, compostos orgânicos voláteis, partículas em suspensão, ponto de orvalho, sensação térmica temperatura e humidade relativa. De forma a proporcionar uma visualização dos dados mais apelativa e intuitiva, os mesmos são apresentados usando um sistema de cores do tipo semáforo. Em função dos valores medidos, estes aparecem ou a vermelho se estiverem muito acima dos

valores padrão anteriormente definidos, amarelo se forem moderados e verde se estiverem dentro do padrão. Uma vez que a maioria das soluções estudadas podem ser acedidas através de qualquer *browser* e plataforma pelo facto de se encontrarem na *Cloud* optou-se por utilizar o serviço *ThingSpeak* para visualizar, armazenar e tratar os dados. Para além disso, os dados também estão armazenados numa base de dados local MySQL e a semelhança do *AirThinx* também possui alertas em tempo real quando algum valor está acima do padrão.

No capítulo que se segue, será feita a definição dos requisitos.

3. Definição de requisitos

Entende-se como requisitos, as características que o sistema a desenvolver deverá cumprir de forma a satisfazer as necessidades da organização bem como dos utilizadores. [14]

3.1. Requisitos funcionais

Os requisitos funcionais descrevem as funcionalidades ou os serviços que se espera que o sistema satisfaça. Na *Tabela 3.1* apresenta-se os requisitos funcionais definidos para o sistema.

Tabela 3.1 – Requisitos funcionais

RF0	O sistema deve ser capaz de efetuar leituras/cálculos dos parâmetros físicos (temperatura e humidade relativa, sensação térmica) e químicos ($PM_{2.5}$, CO_2, VOC_s, ponto de orvalho) pretendidos
RF1	O sistema deve guardar os valores lidos na base de dados *MySQL*
RF2	O sistema deve enviar os dados para a plataforma *ThingSpeak*
RF3	O sistema deve apresentar os dados sob a forma de semáforo, em função dos valores lidos, apresentar as cores: verde, amarelo ou vermelho
RF4	O sistema deve gerar alertas por email caso os valores lidos excedam os parâmetros de referência

3.2. Requisitos não funcionais

Os requisitos não funcionais definem as restrições do sistema relativas a tempo de resposta, capacidade de armazenamento, fiabilidade, usabilidade, escalabilidade, dependência, standards, segurança, entre outros. Na *Tabela 3.2* apresenta-se os requisitos não funcionais definidos para o sistema.

Tabela 3.2 - Requisitos não funcionais

RNF0	O sistema deve ser de fácil compreensão e utilização
RNF1	O sistema deve ser visualmente apelativo
RNF2	O sistema deve efetuar as medições em tempo real
RNF3	O sistema deve estar ligado ininterruptamente a rede
RNF4	O sistema proceder ao envio de notificações em tempo real
RFN5	O sistema deve estar preparado para eventuais futuras expansões

No capítulo que se segue será feita a apresentação das tecnologias a serem utilizadas no desenvolvimento do presente projeto.

4. Implementação

Este capítulo descreve a implementação do projeto. Será feita a apresentação das tecnologias utilizadas, as arquiteturas do *hardware* e *software* bem como o protótipo desenvolvido.

4.1. Tecnologias

O ***MySQL*** é um sistema de gestão de base de dados *open source* que utiliza a linguagem SQL. Neste projeto é utilizado para o armazenamento do histórico das medições.

O ***Arduino (software)*** é um ambiente de desenvolvimento integrado, do inglês *Integrated Development Environment* - IDE utilizado para escrever e carregar programas em C++ para o microcontrolador Arduino. A sua função no corrente projeto é de receber os dados provenientes dos sensores e encaminhá-los para a aplicação c# desenvolvida no *Visual Studio.*

O ***ThingSpeak*** é um serviço da *Web* gratuito que permite ler e armazenar dados do sensor na *Cloud* de forma a desenvolver aplicações relacionadas com a *Internet of Things*. O serviço *Web* do *ThingSpeak* fornece aplicações que permitem analisar e visualizar seus dados no MATLAB e depois proceder ao tratamento dos mesmos. Os dados do sensor podem ser enviados para o *ThingSpeak* a partir do *Arduino, Raspberry Pi, BeagleBone Black* e outros *hardwares.*

A linguagem ***C#*** é uma linguagem de programação orientada a objetos desenvolvida pela *Microsoft* que faz parte da plataforma *.Net*.

O ***Virtual Network Computing*\ - VNC** é um *software* que permite o acesso remoto a um computador por outro que estejam ligados a mesma rede.

O ***Visual Studio*** é um IDE da *Microsoft* utilizado com maior frequência no desenvolvimento de software para o *.Net Framework* e algumas linguagens como: C, C++, C# e J#.

O ***LattePanda*** que é um minicomputador com o sistema operativo *Windows* 10 e Arduino Leonardo embebido fabricado pela empresa *DFRobots*. É possível consultar a sua arquitetura do *hardware* no anexo *A 6.1*

4.2. Arquitetura do hardware

Na *Figura 4.1* apresenta-se a arquitetura do *hardware* que é composta por um nó sensor principal.

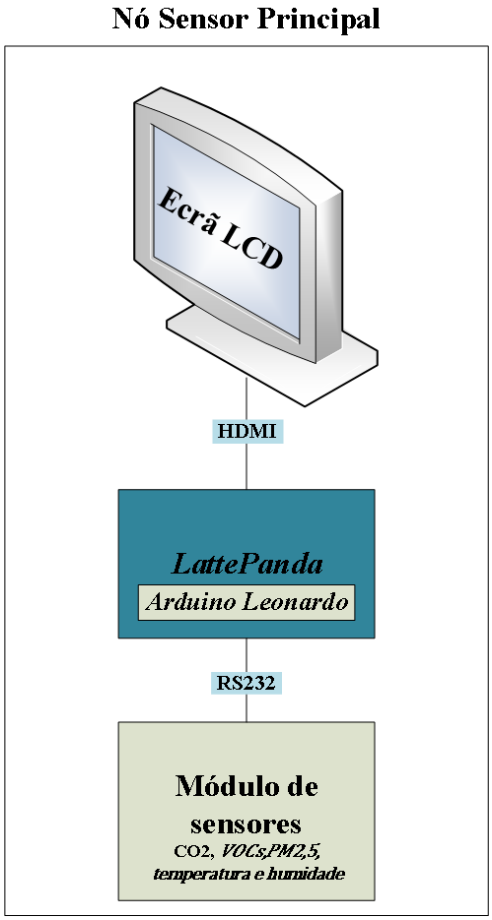


Figura 4.1 – Arquitetura do hardware

O **nó sensor principal** é composto por:

- Um ecrã LCD;
- O *LattePanda* é responsável pela comunicação com o modulo de sensores através do Arduino e pela aplicação Windows desenvolvida.
- Um módulo de sensores constituído por sensores de Dióxido de Carbono - CO_2, Compostos Orgânicos Voláteis - VOC,Partículas em Suspensão - PM, Temperatura e Humidade Relativa.

Na *Figura 4.2* é possível ver o módulo de sensores que foi adquirido ao laboratório de robótica do IPG. Nas secções seguintes é feita uma apresentação de cada sensor que compõe o referido módulo.

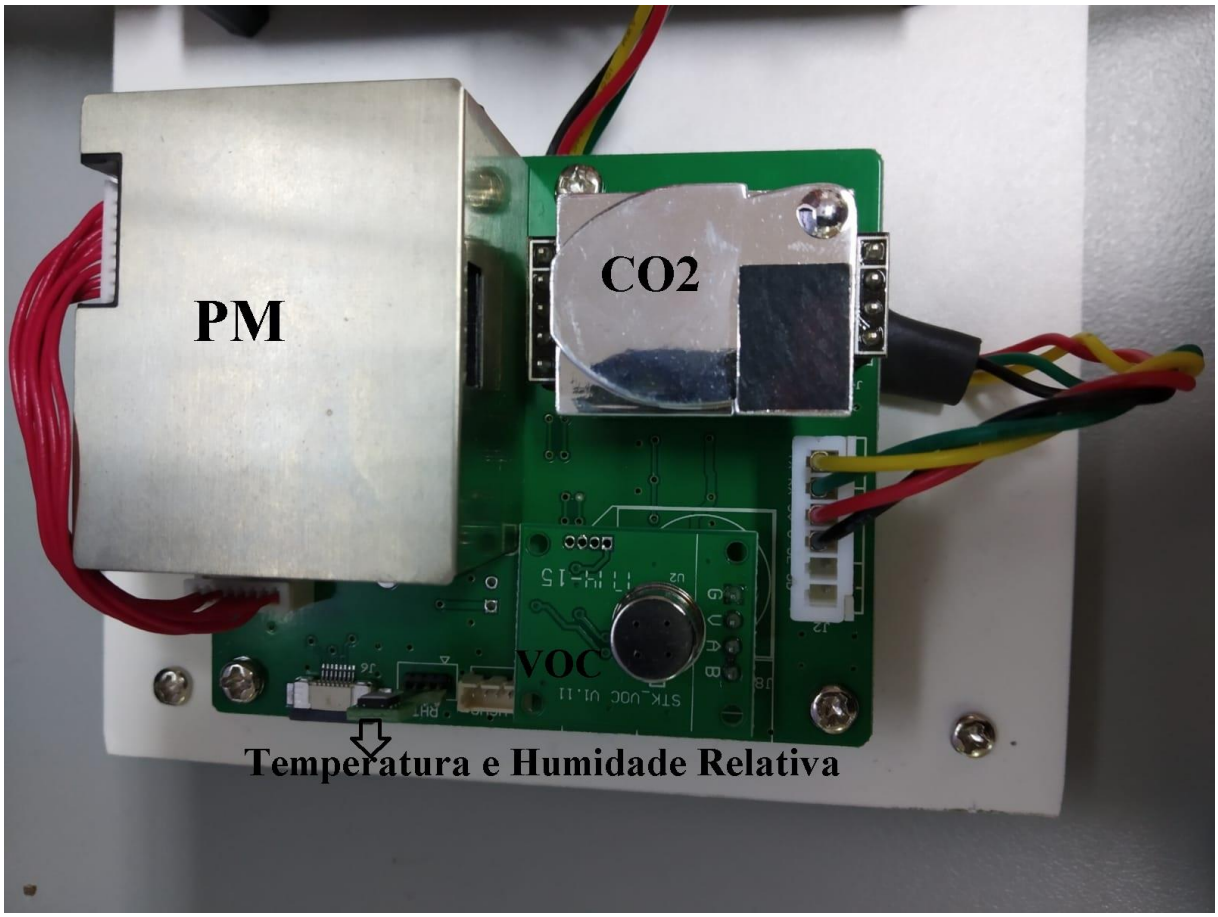


Figura 4.2 - Módulo de Sensores

4.2.1. Sensor de Partículas em Suspensão - $PM_{2.5}$

É um sensor para detetar a concentração de partículas em suspensão existentes no ar. Possui uma ventoinha que atrai o ar para o seu interior para que o mesmo seja analisado através da espectrometria de difração laser[9] e de um conjunto de algoritmos.

[9] Trata-se de uma técnica que permite analisar a distribuição de tamanhos de partícula de uma amostra. Esta técnica baseia-se na medição da variação angular na intensidade da luz difundida à medida que um feixe de laser interage com as partículas dispersas da amostra. [24]

4.2.2. Sensor de Dióxido de carbono - CO_2

Trata-se do sensor que permite medir a concentração de dióxido de carbono presente no ar através da tecnologia de Infravermelho Não Dispersivo do inglês *Non-Dispersive Infrared* – NDIR[10].

4.2.3. Sensor de Compostos Orgânicos Volateis - VOC

Trata-se do sensor de Compostos Orgânicos Voláteis do inglês *Volatile Organic Compounds* – VOC. Este permite a deteção das concentrações de formaldeído, benzeno, monóxido de carbono, amoníaco, hidrogénio, álcool e fumo de cigarro e entre outros sem a distinção dos mesmos. A deteção é feita através do princípio do semicondutor[11].

4.2.4. Sensor de temperatura e humidade relativa

Este sensor efetua medição da temperatura no ar e da humidade relativa correspondente. A medição é efetuada usando condensadores que são conhecidos pela capacidade de armazenarem energia.

4.2.5. Sensor virtual para obter o ponto de orvalho

Este sensor tem o nome de sensor virtual pois o valor do ponto de orvalho é calculado através temperatura e da humidade relativa. Esse cálculo é feito através da equação *(1).*

$$PO = c_1 * (log\left(\frac{H}{100}\right) + ((c_2 * T)/(c_1 + T)))/(c_2 - log\left(\frac{H}{100}\right) - ((c_2 * T)/(c_1 + T)))) \quad (1)$$

Onde:

PO – Ponto de Orvalho

$c_1 = 243.04$

$c_2 = 17.625$

[10] Esta tecnologia utiliza a capacidade do gás de absorver a radiação infravermelha. As amostras que entram no sensor são expostas à luz infravermelha e a concentração de CO2 é calculada em função da quantidade deluz absorvida.

[11] Este princípio consiste na deteção da concentração dos gases através de uma reação química que ocorre quando os mesmos entram em contacto com o sensor. O material geralmente utilizado nos sensores que utilizam esse princípio é o dióxido de estanho. Este tem uma resistência típica de 50 *kiloohms* e tende a baixar quando entra e em contacto com determinados gases. Assim sendo a concentração é medida através da relação entre o valor normal da resistência e o valor após o contacto. [25]

H – Humidade Relativa

T – Temperatura

Fonte: [15]

4.2.6. Sensor virtual para obter a sensação térmica

De modo semelhante ao ponto de orvalho, a sensação térmica é calculada através da equação *(2)*.

$$ST = (c_1 + c_2T + c_3H + c_4TH + c_5T^2 + c_6H^2 + c_7T^2H + c_8TH^2 + c_9T^2H^2 - 32) * \frac{5}{9} \quad (2)$$

Onde:

$ST -$ Sensação Térmica

$c_1 = -42.379$ $c_2 = 2.04901523$ $c_3 = 10.14333127$

$c_4 = -0.22475541$ $c_5 = -0.00683783$ $c_6 = -0.05481717$

$c_7 = 0.00122874$ $c_7 = 0.00122874$ $c_8 = 0.00085282$ $c_9 = 0.00000199$

T – Temperatura

H – Humidade Relativa

Fonte: [15]

Na tabela que se segue, apresenta-se as principais características físico-químicas e elétricas do módulo de sensores de modo a complementar a descrição feita anteriormente. Entre essas características estão a gama de valores disponível, resolução, tempo de resposta, tempo de vida, corrente suportada, dimensões entre outros.

Tabela 4.1- Tabela com as características do módulo de sensores

Fonte: Laboratório de Robótica do IPG

Working condition	Working temperature range	**-10 ~ 50℃**
	Storage temperature range	**-20 ~ 60℃**
	Working humidity range	**30%~80% RH (non-condensing)**
	Storage humidity range	**0~95%RH**
PM2.5 measurement	Principle	**laser scattering technology**
	Measurement range	**0 ~ 1000μg/m³**
	Measurement accuracy	**≤ 100 μg/m³: ±15μg/m³**
		> 100 μg/m³: ±15% of reading, (25℃±2℃, 50±10%RH, TSI8530，cigarette，GBT18801-2015,Temperature influence coefficient: 0.5%/℃ ~ 1%/℃ or 0.5μg/m³/℃ ~ 1μg/m³/℃ whichever is greater)
	Resolution	**1 μg/m³**
	Responding time	**1 second**
	Sampling way	**Fan, 0.6CFM**
	Life span	**Under ambient temperature and pressure, in the condition of continuous use, lifespan is 22000 hours. Lifespan could be prolonged by controlling working time interval of the optical source.**
CO2 measurement	Principle	**NDIR**
	Measurement range	**400-2000ppm**
	Measurement accuracy	**±(50ppm+5% of reading), auto-calibration within temperature and concentration range**
	Responding time (warm up)	**< 120s**
	Resolution	**1ppm**
	Sampling way	**Diffusion**
	lifespan	**8-10 years**
VOC measurement	Principle	**semiconductor**
	Measurement range	**0-3level**
	Sampling way	**Diffusion**
Temperature	Range	**-10~50℃**
	Resolution	**0.1℃**
	Accuracy	**±1.0℃**
Relative humidity	Range	**30~80%RH**
	Resolution	**0.1%RH**
	Accuracy	**±5%RH**
Electric property	Power supply	**5.0V±0.1VDC, ripple wave:<50mW**
	Working current	**<300mA (VOC:<80mA/CO2:<120mA/PM2.5:<80mA)**
	Standby consumption	**≤1.5W**
	Warming-up time	**120 seconds**
	Communication interface	**UART_TTL (3.3V TTL), I2C (3.3V TTL)**
Structure	Dimension	**72*61*25.3mm**

4.3. Arquitetura do *software*

Nesta secção será apresentada a arquitetura do *software* que se trata da implementação do módulo sensor principal anteriormente apresentado.

Na *Figura 6.3* apresenta-se a arquitetura do *software* que é constituída por oito módulos.

O módulo do ***AirQuality.ino***[12] é responsável pela leitura dos valores fornecidos pelo módulo de sensores e encaminhamento dos mesmos para o **AirQuality** (App Windows desenvolvida no *Visual Studio*) através de uma *SoftwareSerial Port*.

Os dados chegam ao **AirQuality** através do módulo ***ArduinoCom.cs,*** posteriormente são armazenados na base de dados *MySQL* (para o armazenamento do histórico de medições) e no *ThingSpeak.* (para a visualização dos gráficos). O módulo responsável pela apresentação dos valores lidos ao público é o ***AppAirQuality.cs***. Quando uma leitura excede o valor padrão é gerada uma notificação por email através do módulo ***SendEmail.cs***.

Todos os módulos da aplicação do Windows são geridos pelo ***FrmPrincipal.cs***.

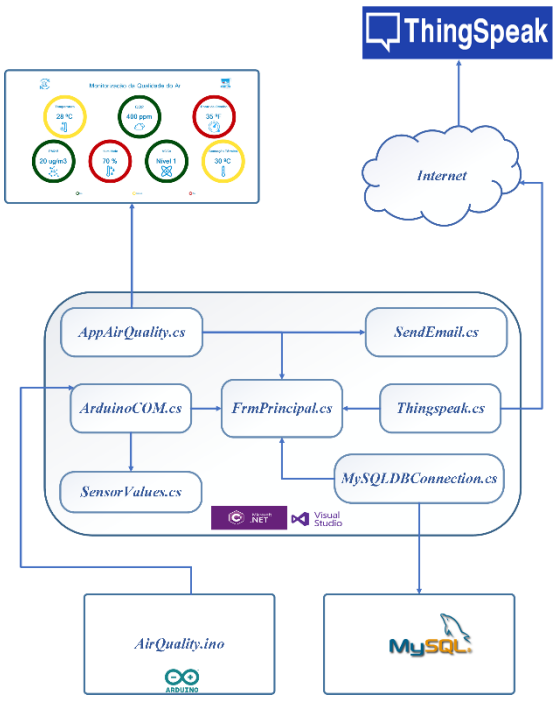


Figura 4.3 – Arquitetura do Software

[12] Código fornecido pelo docente orientador

4.3.1. Módulo AirQuality.ino

Primeiro o módulo envia um código para aplicação *AirQuality*, aguarda pela chegada do mesmo código do *AirQuality* e verifica se coincidem através da seguinte função:

```
void conectWithVisual(){
  String code = "";
  while (code != CODE) {
    while (Serial.available() <= 0) {
      Serial.println(CODE);
      delay(300);
    }
    code = Serial.readString();
}
```

A leitura dos valores é efetuada através do seguinte trecho de código:

```
byte readCmd[] = {0x11, 0x02, 0x01, 0x00, 0xEC};
mySerial.write(readCmd, sizeof(readCmd));
```

O pacote que chega do módulo dos sensores tem o comprimento de 20 bytes e o seguinte formato:

```
16 11 01 aa aa bb bb cc cc dd dd ee ee rr rr rr rr ss ss xx
```

Em que:

- Os três primeiros *bytes* fazem parte do cabeçalho e são fixos;
- Do **aa** ao **ee** correspondem aos valores dos sensores;
- Os **rr** são reservados;
- Os **ss** correspondem ao estado do módulo de sensores
- O **xx** é o *checksum.*

Para a leitura do pacote que chega dos sensores apresentada no código que se segue, primeiro verifica a disponibilidade da porta serie depois guarda no *buffer*.

```
int i=0;
  if (mySerial.available()) {
    delay(100);
    while(mySerial.available() && i < 20) {
      buffer[i++] = mySerial.read();
    }
  }
```

Posteriormente os dados armazenados no *buffer* são processados da seguinte forma:

```
void parceRawData() {
  co2 = buffer[3] * 256 +  buffer[4];
  voc = buffer[5] * 256 +  buffer[6];
  rh = (buffer[7] * 256 +  buffer[8]) / 10.0;
  t = (buffer[9] * 256 +  buffer[10] - 500) / 10.0;
  pm25 = buffer[11] * 256 +  buffer[12];
  state = buffer[17];
    dp = getDewPoint(rh, t);
  hi = getHeatIndex(rh, t);
}
```

Para além dos valores lidos pelos sensores, existem dois que são calculados através da temperatura e da humidade relativa. Tratam-se do ponto de orvalho e a sensação térmica e são obtidos através dos seguintes algoritmos:

- Cálculo do ponto de orvalho

```
float getDewPoint(float h, float t) {
  t = c2f(t);
  return (243.04*(log(h/100)+((17.625*t)/(243.04+t)))/(17.625-log(h/100)-
((17.625*t)/(243.04+t))));
}
```

- Cálculo da sensação térmica:

```
float getHeatIndex(float h, float t) {
  if ((t <= 26.66) || (h <= 40))
    return t;
  else
    return ((-42.379+(2.04901523*t)+(10.14333127*h)-(0.22475541*t*h)-
(0.00683783*t*t)-(0.05481717*h*h)+(0.00122874*t*t*h)+(0.00085282*t*h*h)-
(0.00000199*t*t*h*h)-32)*5/9);
}
```

Fonte: [16]

- Conversão de *celsius* para *fahrenheit:*

```
float c2f(float c) {
  return (c * (9.0/5.0) + 32);
}
```

Depois de lidos e tratados, os valores são enviados para a aplicação *AirQuality* da seguinte forma

```
void sendSensorDataToVisual() {
  Serial.print(co2);
  Serial.print(" ");
  Serial.print(voc);
  Serial.print(" ");
  Serial.print((int)(rh*100));
  Serial.print(" ");
  Serial.print((int)(t*100));
  Serial.print(" ");
  Serial.print(pm25);
  Serial.print(" ");
  Serial.print((int)(hi*100));
  Serial.print(" ");
  Serial.print((int)(dp*100));
  Serial.println("");
}
```

A seguir apresenta-se o diagrama de classes da aplicação *C#* onde é possível visualizar todas as classes que o compõem, bem como as ligações entre elas.

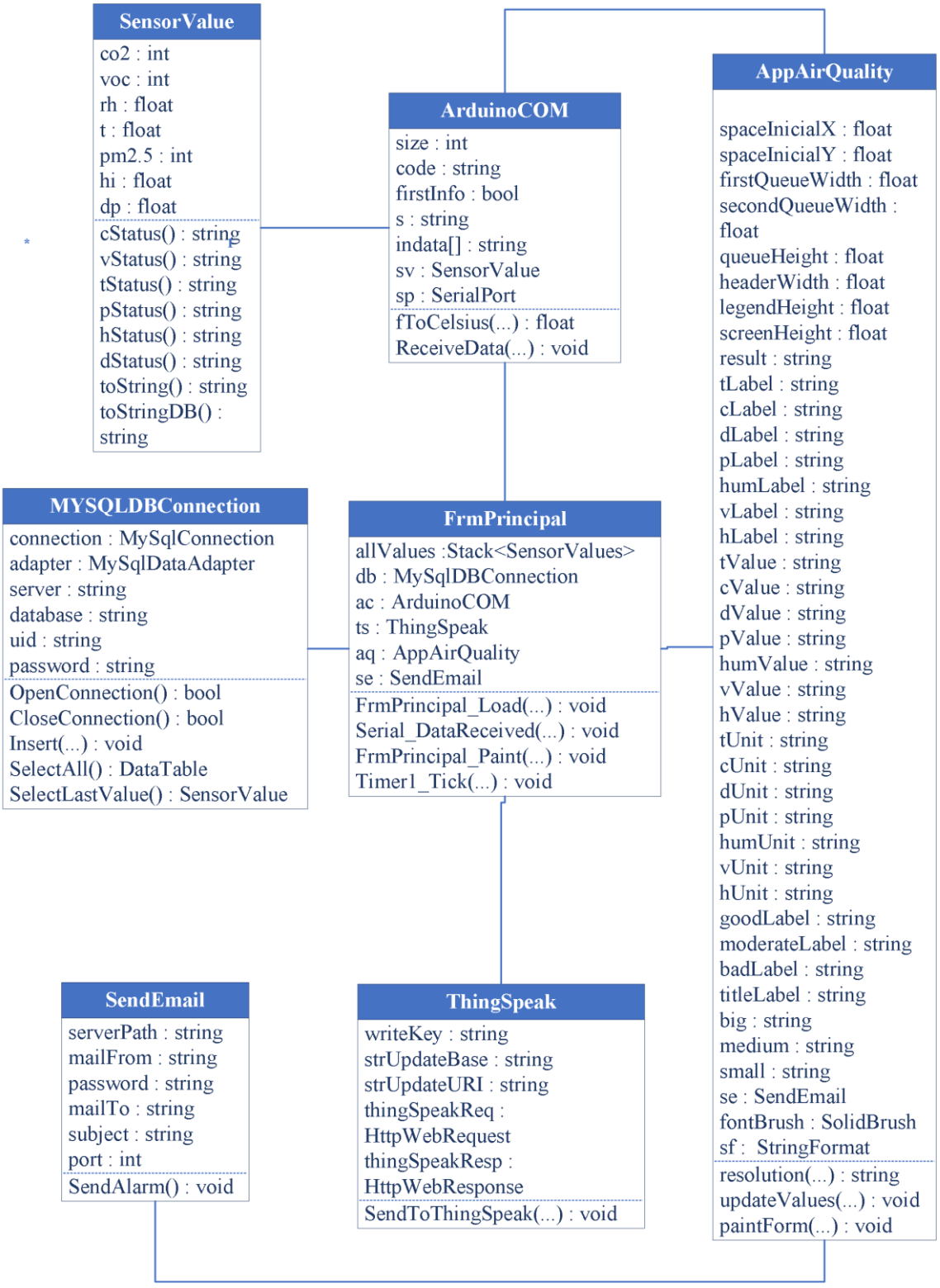


Figura 4.4 – Diagrama de classes da aplicação C#

4.3.2. Modulo *FrmPrincipal*

Como já foi referido, esta classe é responsável pela gestão das restantes classes. Possui 4 eventos responsáveis pelo funcionamento do sistema:

- *FrmPrincipal_Load* - cada vez que é gerado, abre a *serial port* e chama os eventos para receção e apresentação dos dados provenientes do *arduino*;
- *serial_DataReceived* – recebe os dados e armazena-os na base de dados MySQL;
- *frmPrincipal_Paint* – chama a classe responsável pela apresentação dos dados no ecrã;
- *timer1_Tick* – de cinco em cinco minutos envia o último valor lido para o *ThingSpeak.*

O código completo desta classe pode ser consultado no Anexo A 6.1.

4.3.3. Modulo *ArduinoCOM*

Esta classe é responsável pela receção dos dados provenientes do *arduino*. Tal como já foi referido anteriormente, os dados chegam sob forma de *string* no seguinte formato: $CO_2 \ VOC \ RH \ T \ PM_{2.5} \ HI \ DP$

Em que:

- CO_2 - Concentração de dióxido de carbono
- VOC - Nível dos compostos orgânicos voláteis
- RH - Humidade relativa
- T - Temperatura
- $PM_{2.5}$ - Nº de a partículas em suspensão
- HI - Sensação térmica
- DP - Ponto de orvalho

Posteriormente é feita a separação dos diferentes valores, conversão para os tipos de dados correspondentes e são armazenados num objeto do tipo *SensorValue* que por sua vez é inserido

numa *Stack*[13].Este processo é repetido de cinco em cinco segundos. O código pode ser consultado no anexo A 6.2.

4.3.4. Modulo *SensorValue*

Esta classe armazena os valores de uma medição e avalia o seu *status*, ou seja, verifica se é **bom, moderado ou Mau.** A seguir apresenta-se o código para o CO_2 de acordo com os valores de referência apresentados no capítulo 2.

```
public string cStatus() {

            string result="";

            if (co2 >= 0 && co2 < 1250)
            {
                result = "good";
            }
            else if (co2 >= 1250 && co2 < 5000)
            {
                result = "moderate";
            }
            else if(co2 >= 5000){
                result = "bad";
            }
            return result;
        }
```

Para os outros parâmetros o procedimento foi semelhante, o código completo pode ser consultado no nexo A 6.3.

4.3.5. Modulo *AppAirQuality*

Trata-se da classe responsável pela parte gráfica do sistema, ou seja, por apresentar os valores lidos no ecrã.

O conceito utilizado foi o semáforo. Em função dos valores lidos, estes são apresentados com as cores verde, amarelo e vermelho respetivamente.

Todos os objetos desenhados no ecrã pertencem a classe *Graphics*. Estes são desenhados em função das dimensões do monitor.

[13] Estrutura de dados do tipo LIFO – *Last In First Out*

É possível obter as dimensões do monitor com o seguinte trecho de código:

```
width = Screen.PrimaryScreen.Bounds.Width;
height = Screen.PrimaryScreen.Bounds.Height;
```

As restantes dimensões que permitem desenhar os elementos do ecrã são calculadas da seguinte forma:

```
spaceInicialY = 0.2F * height;
spaceInicialX = 0.1F * width;
queueHeight = 0.35F * height;
firstQueueWidth = 0.33F * width;
secondQueueWidth = 0.25F * width;
headerWidth = width;
legendHeight = 0.10F * height;
screenHeight = height;
```

Foram definidas três categorias para resolução do ecrã: grande, média e pequena. Esse cálculo é feito com base nas dimensões do ecrã como se pode ver na função que se segue:

```
private string Resolution(int w, int h)
        {
            string result = "";
            if ((w > 1024) && (h >= 1080))
            {
                result = big;
            }
            else if ((w > 800 || w <= 1024) && (h >= 600 || h < 1080))
            {
                result = medium;
            }
            else if ((w >= 0 || w <= 800) && (h >= 0 || h <= 600))
            {
                result = small;
            }
            return result;
        }
```

Em função da resolução obtida define-se um tamanho das letras com o seguinte código:

```
if (Result.Equals(small) || Result.Equals(medium))
{
      titleFont = new Font("Microsoft Sans Serif", 50 *
0.5F,FontStyle.Bold);
      labelsFont = new Font("Microsoft Sans Serif", 20 *
0.5F,FontStyle.Bold);
      footerFont = new Font("Microsoft Sans Serif", 20 *
0.5F,FontStyle.Bold);

}else if (Result.Equals(big))
{
      titleFont = new Font("Microsoft Sans Serif", 50, FontStyle.Bold);
      labelsFont = new Font("Microsoft Sans Serif", 20, FontStyle.Bold);
      footerFont = new Font("Microsoft Sans Serif", 20, FontStyle.Bold);
}
```

Para escrever o título por exemplo, utiliza-se o seguinte código:

```
SizeF size = g.MeasureString(titleLabel, titleFont);
g.DrawString(titleLabel, titleFont, fontBrush, (headerWidth - size.Width) /
2, (screenHeight * 0.2F - size.Height) / 2, sf);
```

Ecrã da aplicação:

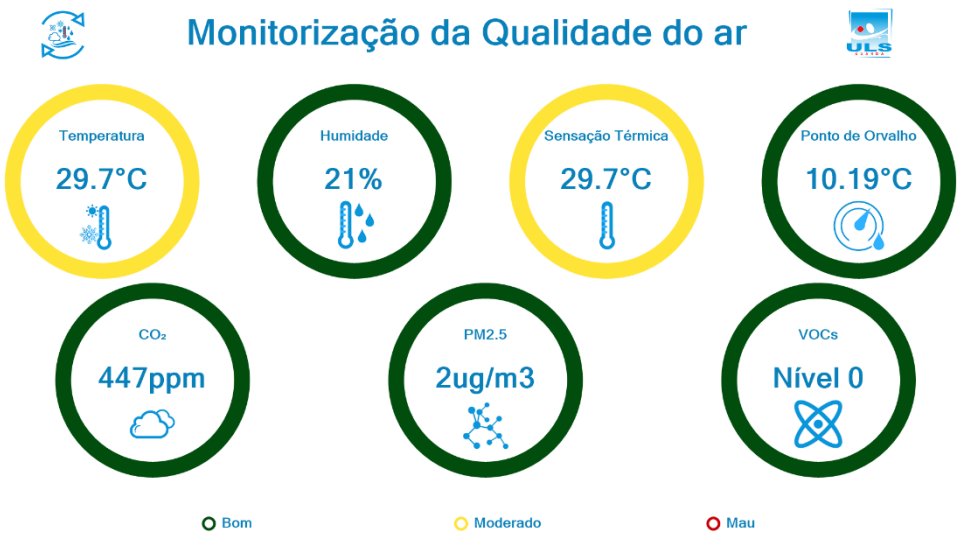


Figura 4.5 - Ecrã da aplicação

O Código completo pode ser consultado no Anexo A 6.4.

4.3.6. Modulo *MySqlDBConnection*

Nesta classe são feitas as configurações necessárias para ligação a base de dados MySQL, e algumas operações como inserção de valores e visualização dos mesmos.

Código para criar a ligação à base de dados:

```
server = "localhost";
database = "airqualitydb";
uid = "root";
password = "jasimaoIPG2018";
connectionString = "SERVER=" + server + ";" + "DATABASE=" +
```

```
database + ";" + "UID=" + uid + ";" + "PASSWORD=" + password + ";";
connection = new MySqlConnection(connectionString);
```

A inserção de um elemento na base de dados é feita através da seguinte função:

```
public void Insert(string s)
{
…      string query = "INSERT INTO sensorvalue
       (co2,voc,rh,t,pm25,hi,dp,date) VALUES" + s;
       if (OpenConnection() == true)
       {
              MySqlCommand cmd = new MySqlCommand(query, connection);
              cmd.ExecuteNonQuery();
              CloseConnection();
       }
}
```

O *Select* de todos os dados guardados na base de dados é feito através da seguinte função:

```
public DataTable SelectAll()
{

              string query = "SELECT * from sensorvalue";

              //open connection
              try
              {
                  OpenConnection();
                  MySqlCommand cmd = new MySqlCommand(query, connection);
                  cmd.ExecuteNonQuery();
                  adapter = new MySqlDataAdapter(cmd);
                  DataTable dt = new DataTable();
                  adapter.Fill(dt);
                  return dt;
              }
              catch (MySqlException ex)
              {
                  throw new ApplicationException(ex.ToString());
              }
              finally
              {
                  CloseConnection();
              }

}
```

O *Select* do último elemento inserido na base de dados é feito através da seguinte função:

```
public SensorValue SelectLastEntry()
        {

              string query = "SELECT * FROM sensorvalue ORDER BY ID DESC
LIMIT 1";

              //open connection
              try
```

```
            {
                OpenConnection();
                //create command and assign the query and connection from
the constructor
                MySqlCommand cmd = new MySqlCommand(query, connection);
                //Execute command
                 cmd.ExecuteNonQuery();
                adapter = new MySqlDataAdapter(cmd);
                DataTable dataTable = new DataTable();
                adapter.Fill(dataTable);

                //SensorValues auxiliar
                SensorValue aux = new SensorValue();
                aux.c = Convert.ToInt32(dataTable.Rows[0][1]);
                aux.v = Convert.ToInt32(dataTable.Rows[0][2]);
                aux.r = Convert.ToInt32(dataTable.Rows[0][3]);
                aux.temp = Convert.ToInt32(dataTable.Rows[0][4]);
                aux.pm = Convert.ToInt32(dataTable.Rows[0][5]);
                aux.h = Convert.ToInt32(dataTable.Rows[0][6]);
                aux.d = Convert.ToInt32(dataTable.Rows[0][7]);
                //return dt.ToString();
                return aux;
            }
            catch (MySqlException ex)
            {
                throw new ApplicationException(ex.ToString());
            }
            finally
            {
                //close connection
                CloseConnection();
            }

        }
```

4.3.7. Modulo *SendEmail*

Trata-se da classe responsável pelo envio de *emails* de alerta usando *Simple Mail Transfer Protocol* – SMTP que se trata de um protocolo utilizado para envio de *emails*. Esses emails são gerados quando algum valor excede os parâmetros pré-definidos como padrão.

A configuração da conta através da qual se pretende enviar os emails é feita com a seguinte função:

```
public SendEmail(string b)
        {
            serverPath = "smtp.office365.com";
            mailFrom = "airqualityulsg@hotmail.com";
            password = "jasimaoIPG2018";
            mailTo = "jana-simao@hotmail.com";
            subject = "AirQuality alert email";
            body = b;
            port = 587;
            mail = new MailMessage();
            server = new SmtpClient(serverPath);
        }
```

A função para o envio de emails:

```
public void SendAlarm()
        {
            try
            {
                MailMessage mail = new MailMessage();
                SmtpClient SmtpServer = new SmtpClient(serverPath);

                mail.From = new MailAddress(mailFrom);
                mail.To.Add(mailTo);
                mail.Subject = subject;
                mail.Body = body;

                SmtpServer.Port = port;
                SmtpServer.Credentials = new
                System.Net.NetworkCredential(mailFrom, password);
                SmtpServer.EnableSsl = true;

                SmtpServer.Send(mail);
                MessageBox.Show("mail Send");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }

        }
```

4.3.8. Modulo *ThingSpeak*

Neste módulo é feito o envio dos dados dos sensores para a plataforma *ThingSpeak*.

Para isso, após a criação de uma conta *thingSpeak* e de um canal, é necessário definir uma chave de leitura, *string* que vai ser utilizada na atualização dos valores recebidos dos sensores com o seguinte código:

```
WRITEKEY = "O0051IMWSCN1L8SB";
            strUpdateBase = "http://api.thingspeak.com/update";
            strUpdateURI = strUpdateBase + "?api_key=" + WRITEKEY;
            webclient = new WebClient();
```

O envio dos dados para a plataforma é feito com a seguinte função :

```
public void sendToThingspeak(Stack<SensorValue> allValues)
        {
            ////send read data to thingSpeak plataform
            try
            {
            strUpdateURI += "&field1=" + allValues.Peek().c;
            strUpdateURI += "&field2=" + allValues.Peek().v;
            strUpdateURI += "&field3=" + allValues.Peek().r;
            strUpdateURI += "&field4=" + allValues.Peek().temp;
            strUpdateURI += "&field5=" + allValues.Peek().pm;
            strUpdateURI += "&field6=" + allValues.Peek().h;
            strUpdateURI += "&field7=" + allValues.Peek().d;

            webclient.UploadString(strUpdateURI, "POST", "");

            }
            catch (WebException ex)
            {
                Console.WriteLine("This program is expected to throw
WebException on successful run." +
                                  "\n\nException Message :" +
ex.Message);
                if (ex.Status == WebExceptionStatus.ProtocolError)
                {
                    Console.WriteLine("Status Code : {0}",
((HttpWebResponse)ex.Response).StatusCode);
                    Console.WriteLine("Status Description : {0}",
((HttpWebResponse)ex.Response).StatusDescription);
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
```

Na *Figura 4.6* apresenta-se os gráficos gerados pelo ThingSpeak.

Figura 4.6 – Gráficos gerados pelo ThingSpeak

4.4. Protótipo

Na *Figura 4.7*apresenta-se o protótipo obtido onde é possível ver o monitor com a apresentação dados dos diferentes sensores, o modulo de sensores e o *LattePanda.*

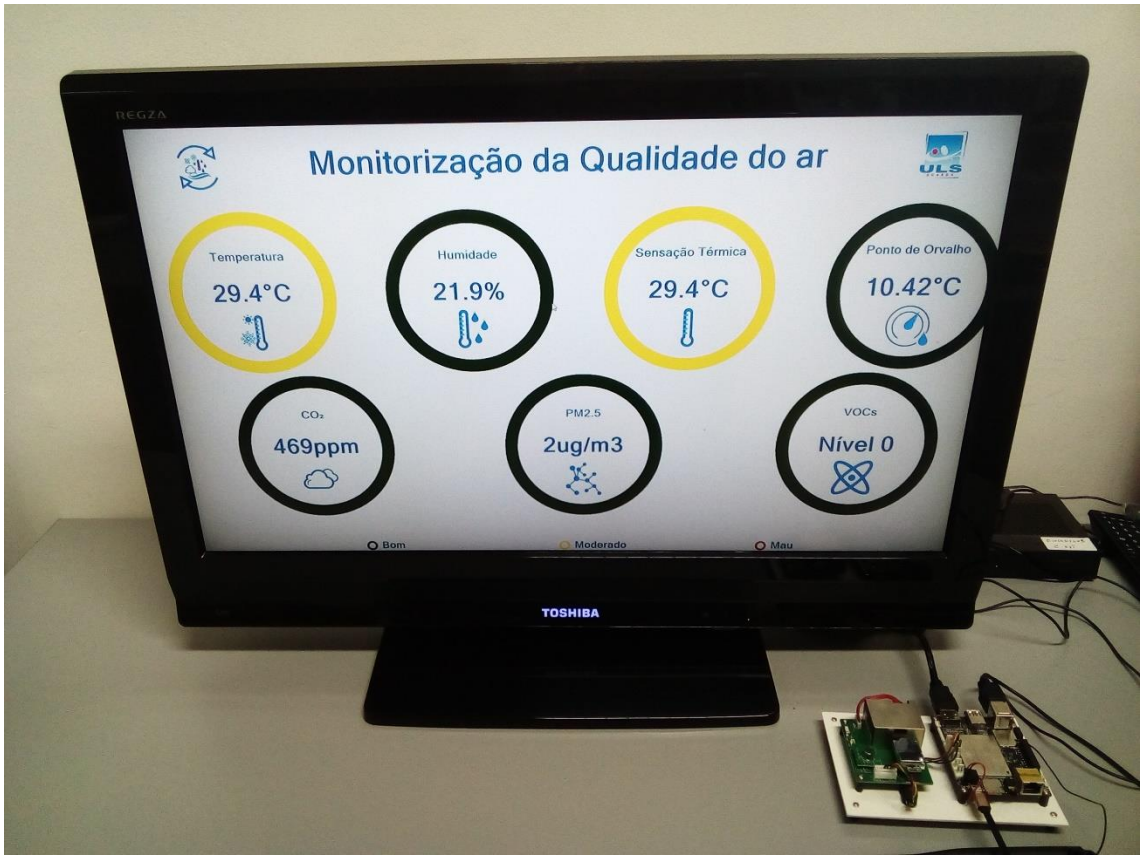


Figura 4.7 – Protótipo obtido

5. Verificação e Validação

Foram realizados alguns testes periódicos ao sistema de forma a verificar o seu correto funcionamento.

- **Teste do Sensor de compostos orgânicos voláteis - VOC:**

Foi colocado um frasco de álcool etílico ao pé dos sensores e o nível aumentou para 3 que corresponde ao nível máximo como se pode verificar na *Figura 5.1*.

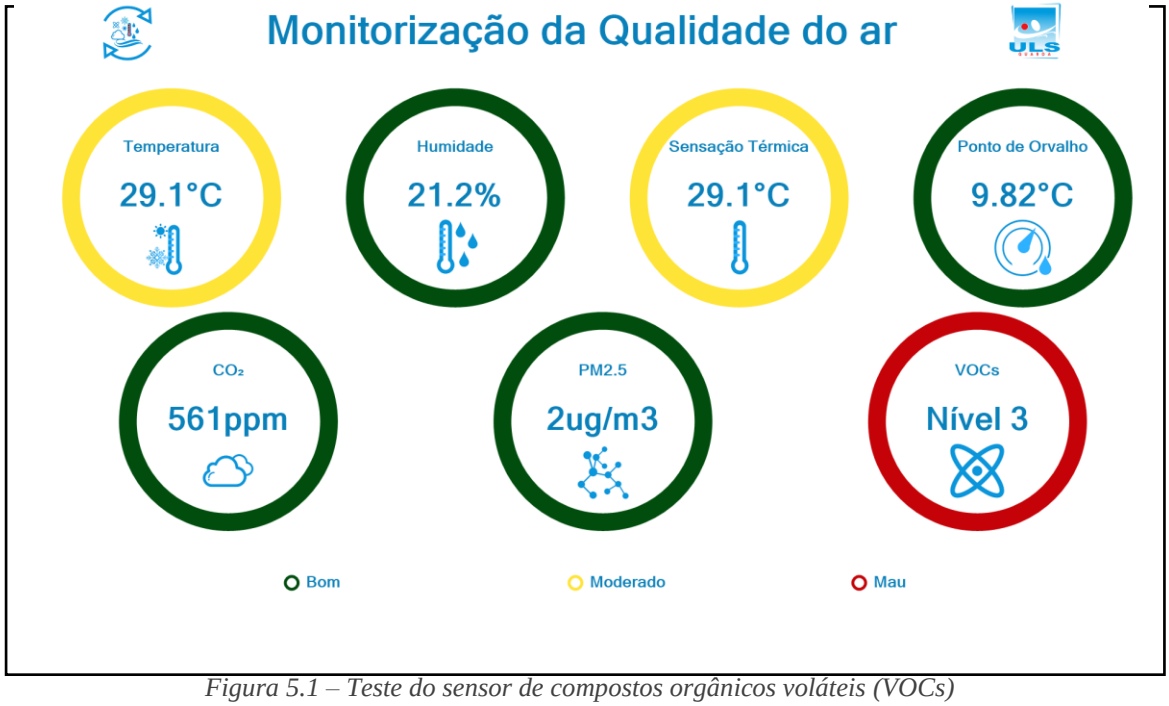


Figura 5.1 – Teste do sensor de compostos orgânicos voláteis (VOCs)

Ainda se verificou que um email de alerta foi corretamente enviado como a figura seguinte demonstra.

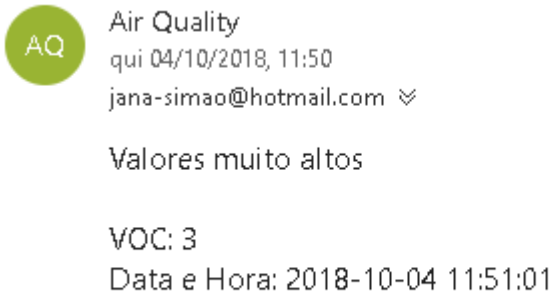


Figura 5.2 – Email de alerta VOC

- **Teste do Sensor de partículas em suspensão:**

Para esse teste colocou-se um pedaço de papel em chamas num pote perto dos sensores e obteve-se o seguinte resultado:

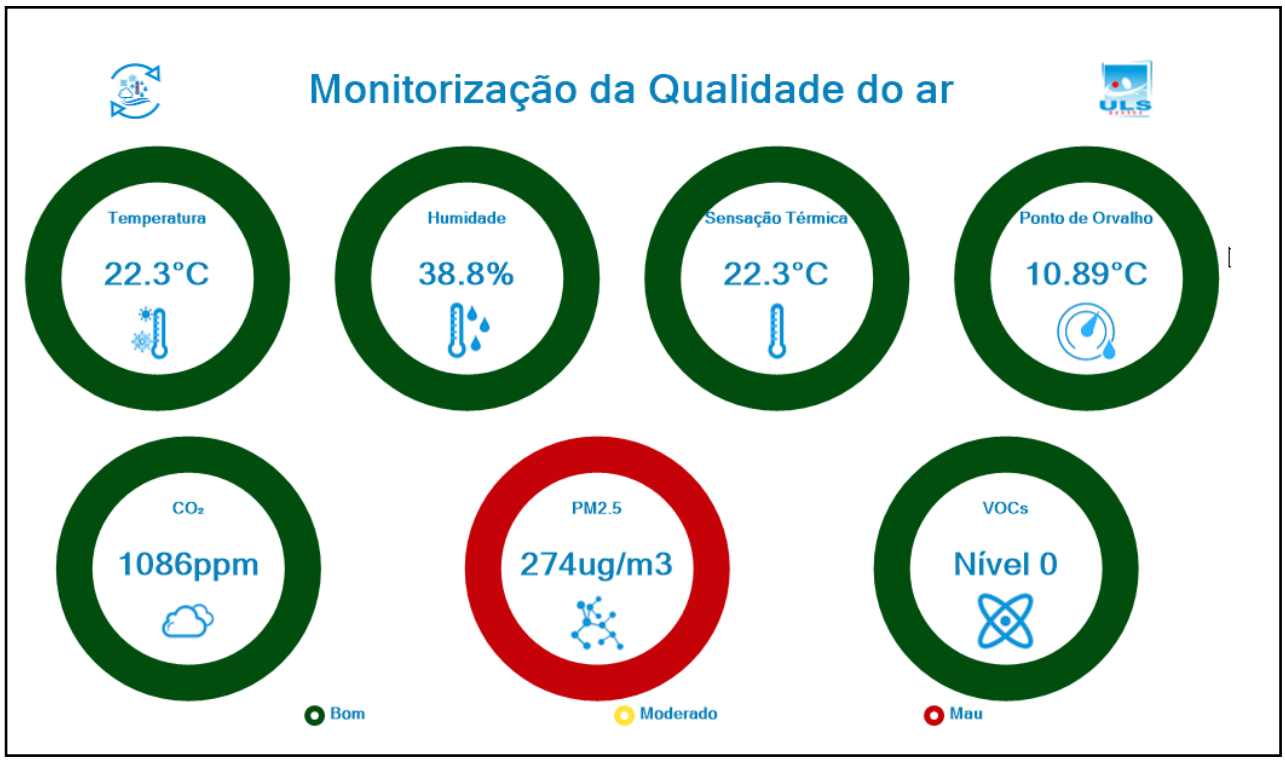


Figura 5.3 – Teste do sensor de Partículas em suspensão (PM2.5)

O email de alerta foi enviado corretamente como se pode ver na figura seguinte:

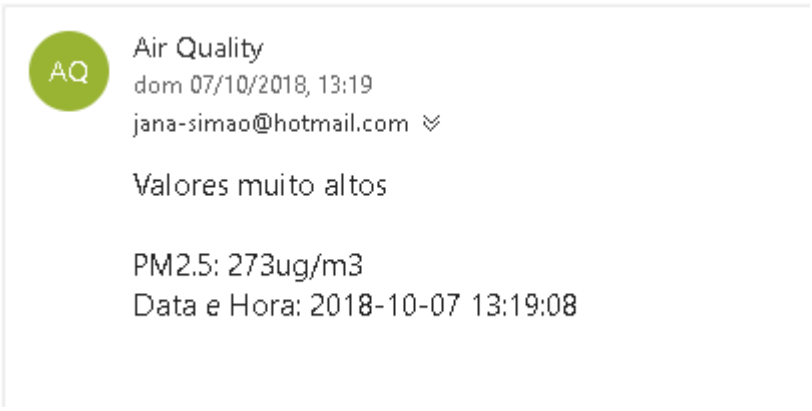
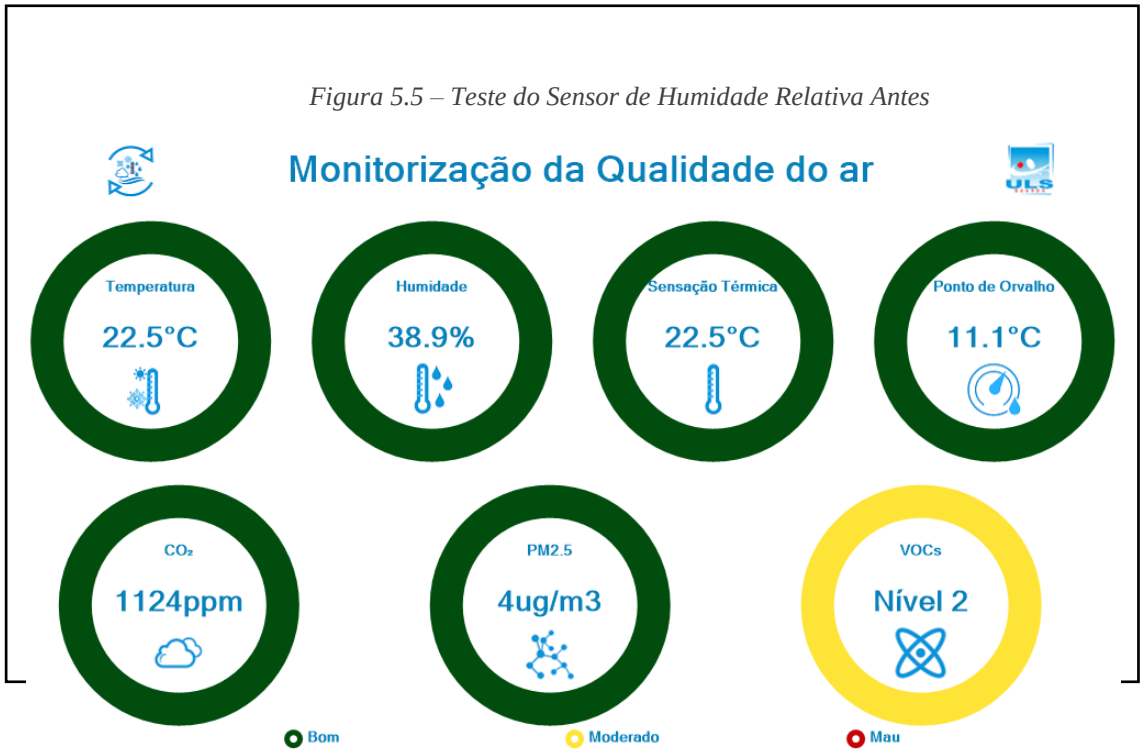


Figura 5.4 – Alerta PM2.5

- **Teste humidade relativa**

Colocou-se um recipiente com água quente perto do modulo de sensores e verificou-se que o valor da humidade relativa aumentou.

Figura 5.5 – Teste do Sensor de Humidade Relativa Antes



Verificou-se um aumento da humidade relativa como é possível ver na figura que se segue.

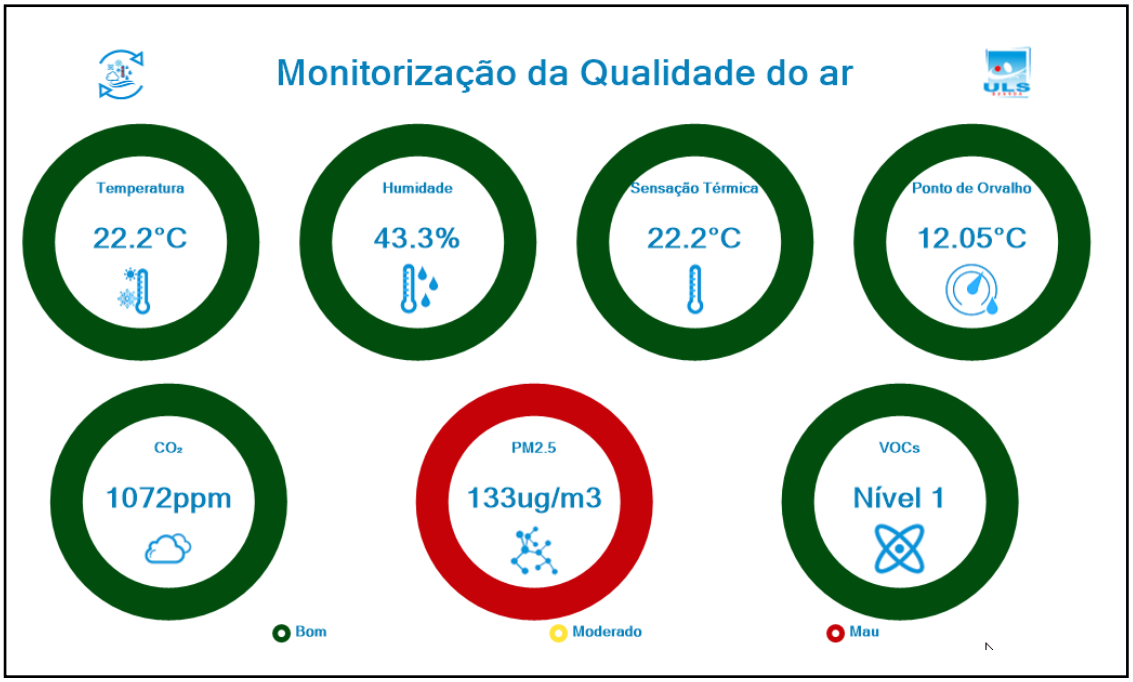


Figura 5.6 - Teste do Sensor de Humidade Relativa Depois

6. Conclusões

É importante realçar a pertinência do desenvolvimento de projetos que permitam melhorar a qualidade de vida, como é o caso da observação da qualidade do ar nas casas, no trabalho, nos hospitais entre outros. Estes tornaram-se um tema muito abordado e a medida que o tempo vai passando têm surgido cada vez mais produtos e projetos dessa área, o que constitui uma mais-valia.

Este projeto que é de grande relevância cumpriu todos os objetivos definidos inicialmente. Obteve-se um primeiro protótipo funcional do sistema de monitorização da qualidade do ar interior, com informações que podem ser visualizadas em qualquer lado através do *ThingSpeak* e acedidas via *Virtual Network Computing* - VNC, aplicação que permite fácil acesso e controle de um computador a partir do outro desde que estejam ligados a mesma rede.

Por se tratar de um projeto piloto, foi desenvolvido um módulo num só espaço físico, podendo o mesmo numa fase posterior ser disseminado por todos os compartimentos do hospital.

A minha permanência no SSTIC-ULSG foi positiva e também concorreu para a realização exitosa desse projeto, pois contribuiu de forma muito positiva para a aquisição, aplicação e enriquecimento dos conhecimentos relativos a qualidade do ar interior, *Internet of Things* – IoT, o serviço *ThingSpeak*, a linguagem de programação C#.

6.1. Trabalho futuro

Como trabalho futuro pretende-se:

- Resolver algumas falhas como por exemplo o fato do sistema ainda não ser capaz de se recuperar sozinho de uma falha de rede, ou de corrente;
- Gerar relatórios periódicos;
- Outros tipos de notificação: SMS, sinais sonoros;
- Ter vários aparelhos semelhantes ao protótipo obtido interligados entre si através do protocolo REST para que seja possível efetuar medição em espaços distintos;
- Saber a localização dos sensores para que as notificações cheguem com informações relativas ao local da ocorrência;
- Fazer o tratamento dos dados armazenados na base de dados e tirar partido dos conhecimentos de inteligência artificial para obter uma previsão de medições futuras;
- Explorar melhor as funcionalidades do *ThingSpeak.*

Referências

[1] ulsguarda, “Cuidados de Saude Hospitalares,” [Online]. Available: http://www.ulsguarda.min-saude.pt/servicos/cuidados-de-saude-hospitalares/csh1/. [Acedido em 30 05 2018].

[2] SERVIÇO DE SISTEMAS E TECNOLOGIAS DA INFORMAÇÃO E COMUNICAÇÕES, “Regulamento Interno do SSTIC,” 2012.

[3] S. Fernandes, “Sistema de Monitorização e de Controlo de Qualidade do Ar,” 2015. [Online]. Available: http://hdl.handle.net/10198/12713. [Acedido em 6 junho 2015].

[4] C. Bridger, “National Public Health Week: The Health Impacts of Indoor Air Quality,” [Online]. Available: https://www.neefusa.org/health/asthma/national-public-health-week-health-impacts-indoor-air-quality. [Acedido em 13 Junho 2018].

[5] World Health Organization (WHO), “WHO guidelines for indoor air quality: selected pollutants,” 2010. [Online]. Available: http://www.euro.who.int/__data/assets/pdf_file/0009/128169/e94535.pdf. [Acedido em 12 Junho 2018].

[6] M. S. Guerreiro Sanguessuga, “Síndroma dos Edifícios Doentes,” 23 Abril 2012. [Online]. Available: https://repositorio.ipl.pt/bitstream/10400.21/1597/5/S%C3%ADndrome%20dos%20edificios%20doentes.pdf. [Acedido em 13 Junho 2018].

[7] World Health Organization - WHO, “Sick building syndrome,” [Online]. Available: https://www.wondermakers.com/Portals/0/docs/Sick%20building%20syndrome%20by%20WHO.pdf. [Acedido em 13 Junho 2018].

[8] World Health Organization, “Clean Care is Safer Care,” [Online]. Available: http://www.who.int/gpsc/country_work/burden_hcai/en/. [Acedido em 14 Junho 2018].

[9] Diário da República, “Diário da República,” 4 Dezembro 2013. [Online]. Available: https://dre.pt/application/conteudo/331868. [Acedido em 12 06 2018].

[10] A. Guballa, D. Dunn e M. Mitchell, “SAFETY HEALTH & WELLBEING,” 8 novembro 2017. [Online]. Available: http://sydney.edu.au/whs/guidelines/others/WHS_GEN_GUI_Indoor%20Air%20Quality%20and%20Thermal%20Comfort.pdf. [Acedido em 12 junho 2018].

[11] X. Yang, L. Yang e J. Zhang, “IEEE Explorer,” 8 Agosto 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8003185/. [Acedido em 11 06 2018].

Fluke, “Fluke 975 AirMeter,” [Online]. Available: http://www.fluke.com/fluke/uken/HVAC-IAQ-Tools/Air-Testers/Fluke-975.htm?PID=56156. [Acedido em 12 Junho 2018].

[13] Airthinx, “Products,” [Online]. Available: https://airthinx.io/products/. [Acedido em 12 Junho 2018].

[14] I. Sommerville, “Chapter 4: Requirements Engineering,” 2 Janeiro 2015. [Online]. Available: https://www.slideshare.net/software-engineering-book/ch4-req-eng. [Acedido em 16 Julho 2018].

[15] Environment Health Perspectives, “Methods to Calculate the Heat Index as an Exposure Metric in Environmental Health Research,” [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3801457/. [Acedido em 15 Julho 2018].

[16] “National Weather Service,” [Online]. Available: https://www.weather.gov/ama/heatindex. [Acedido em 1 agosto 2018].

[17] Lattepanda, “Lattepanda Docs,” [Online]. Available: https://www.lattepanda.com/lattepanda-docs#. [Acedido em 1 agosto 2018].

[18] G. Marques e R. Pitarma, “2016 11th Iberian Conference on Information Systems and Technologies (CISTI),” 28 Julho 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7521375/. [Acedido em 11 Junho 2018].

[19] elprocu, “elprocu,” [Online]. Available: https://www.elprocus.com/what-is-zigbee-technology-architecture-and-its-applications/. [Acedido em 11 Junho 2018].

[20] Wolfsense, “directsense iaq indoor air quality monitor,” [Online]. Available: http://www.wolfsense.com/directsense-iaq-indoor-air-quality-monitor.html. [Acedido em 12 Junho 2018].

[21] Scrum Portugal, [Online]. Available: http://www.scrumportugal.pt/metodologia-scrum/. [Acedido em 16 Julho 2018].

[22] K. Schwaber e J. Sutherland, “Scrumguides,” Novembro 2017. [Online]. Available: http://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Portuguese-European.pdf. [Acedido em 16 Julho 2018].

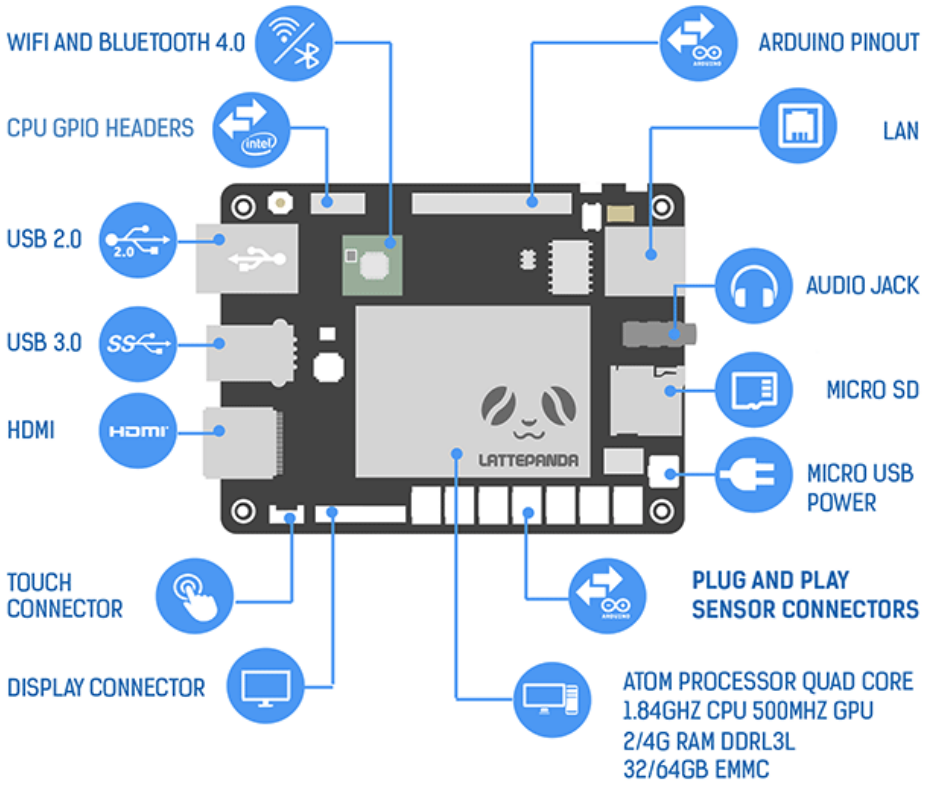
[23] Such Engenharia, “Controlo da Qualidade do Ambiente Interior,” 2014.

[24] Instituto Pedro Nunes, “ipn,” [Online]. Available: https://www.ipn.pt/laboratorio/LEDMAT/ensaio/13. [Acedido em 1 agosto 2018].

[25] [Online]. Available: http://www.figarosensor.com/products/general.pdf. [Acedido em 15 Julho 2018].

Anexos

A 6.1 Arquitetura do *hardware* do *lattepanda*



Fonte: [17]

FrmPrincipal.cs

```
using System;
using System.IO.Ports;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Drawing;
namespace AirQuality
{
    public partial class FrmPrincipal : Form
    {
        SensorValues init = new SensorValues();
        //save all sensor data LIFO
        Stack<SensorValues> allValues = new Stack<SensorValues>();
        MySqlDBConnection db = new MySqlDBConnection();
        ArduinoCOM ac;
        ThingSpeak ts;
        AppAirQuality aq;
        SendEmail se;
        Color blue;

        public FrmPrincipal()
        {
            blue = Color.FromArgb(8, 128, 186);
            InitializeComponent();
            this.Width = Screen.PrimaryScreen.Bounds.Width;
            this.Height = Screen.PrimaryScreen.Bounds.Height;
            this.ClientSize = new System.Drawing.Size(this.Width,
this.Height);

            ac = new ArduinoCOM();
            ts = new ThingSpeak();
            aq = new AppAirQuality(this.Width, this.Height);
            se = new SendEmail();
            allValues.Push(init);
            timer1.Start();
        }
        public void FrmPrincipal_Load(object sender, EventArgs e)
        {
            serial.Open();
      serial.DataReceived +=
      new SerialDataReceivedEventHandler(serial_DataReceived);
            this.Paint += new PaintEventHandler(this.frmPrincipal_Paint);
        }
        //Receive the data from arduino and save on MySQL database
        public void serial_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {
            ac.ReceiveData(sender, allValues, db);
            db.Insert(allValues.Peek().ToStringDB());
            //se.SendAlarm
            this.Invalidate();
        }
        public void frmPrincipal_Paint(object sender, PaintEventArgs e)
        {
```

```
        e.Graphics.SmoothingMode =
        System.Drawing.Drawing2D.SmoothingMode.AntiAlias;  //make the
        graphics better
              aq.paintForm(sender, e, allValues);
          }
          //Send the values to thingSpeak
          private void timer1_Tick(object sender, EventArgs e)
          {
              ts.sendToThingspeak(allValues);
          }
    }
}
```

A 6.2 ArduinoCOM.cs

```
using System;
using System.Collections.Generic;
using System.IO.Ports;
using System.Threading;

namespace AirQuality
{
    class ArduinoCOM
    {
        private readonly int size;
        private readonly string code;
        private bool firstInfo;
        private String s;
        private String[] indata;
        private SensorValues sv;
        private int[] parsedValues;
        private SerialPort sp;

        public ArduinoCOM()
        {
            size = 7;
            code = "AQ_ULSG";
            firstInfo = false;
            sv = new SensorValues();
            s = "";
        }
        public float fToCelsius(float f)
        {
            return (float)Math.Round((double)5F/9F*(f - 32F),2);
        }

        public void ReceiveData(Object sender, Stack<SensorValues>
allValues, MySqlDBConnection db)
        {

            Start: Thread.Sleep(5000); //the same on arduino program
            string[] separatingChars = { " " };
            sp = (SerialPort)sender;
            //send handshake code to visual studio
            if (firstInfo == false)
            {
                sp.Write(code);
                firstInfo = true;
            }

            //read serial data from arduino program
            s = sp.ReadExisting();
            indata = s.Split(separatingChars,
StringSplitOptions.RemoveEmptyEntries);
            parsedValues = new int[size];

            //verify the length of the packet sent by arduino
            if (indata.Length != size)
            {
                Array.Clear(indata, 0, indata.Length);
                goto Start;
            }
```

```
            //parse from string[] to int[]
            for (int i = 0; i < indata.Length; i++)
            {
                int.TryParse(indata[i], out parsedValues[i]);
            }

            Array.Clear(indata, 0, indata.Length);

            sv.c = parsedValues[0];
            sv.v = parsedValues[1];
            sv.r = Convert.ToSingle(parsedValues[2]) / 100;
            sv.temp = Convert.ToSingle(parsedValues[3]) / 100;
            sv.pm = parsedValues[4];
            sv.h = Convert.ToSingle(parsedValues[5]) / 100;
            sv.d = fToCelsius(Convert.ToSingle(parsedValues[6]) / 100);
            sv.dateTime = DateTime.Now;

            Array.Clear(parsedValues, 0, parsedValues.Length);
            allValues.Push(sv); //save last read data on sensorvalues stack
            sv = new SensorValues();

            Console.WriteLine("\n Sensor Values: ");
            Console.Write(allValues.Peek().ToString()); //print last read
data
            Console.WriteLine();

        }

    }

}
```

A 6.3 SensorValue.cs

```
using System;

namespace AirQuality
{
    class SensorValue
    {
        //variables definition
        private int co2;
        private int voc;
        private float rh;
        private float t;
        private int pm25;
        private float hi;
        private float dp;
        DateTime date;

        public SensorValue()
        {
            co2 = 400;
            voc = 1;
            rh = 70;
            t = 28;
            pm25 = 20;
            hi = 30;
            dp = 35;
            date = DateTime.Now;
        }
        //gets and sets
        public int c
        {

            get { return co2; }
            set { co2 = value; }
        }
        public int v
        {

            get { return voc; }
            set { voc = value; }
        }
        public float r
        {
            get { return rh; }
            set { rh = value; }

        }

        public float temp
        {
            get { return t; }
            set { t = value; }
        }
        public int pm
        {
            get { return pm25; }
            set { pm25 = value; }
```

```
        }
        public float h
        {
            get { return hi; }
            set { hi = value; }
        }
public float d
        {
            get { return dp; }
            set { dp = value; }
        }
        public DateTime dateTime {
            get { return date; }
            set { date = value; }
        }

        // this functions validate the data of the sensors
        public string cStatus() {

            string result="";

            if (co2 >= 0 && co2 < 1250)
            {
                result = "good";
            }
            else if (co2 >= 1250 && co2 < 5000)
            {
                result = "moderate";
            }
            else if(co2 >= 5000){
                result = "bad";
            }
            return result;
        }

        public string vStatus()
        {
            string result = "";
            if (voc == 0 || voc == 1)
            {
                result = "good";
            }
            else if (voc == 2)
            {
                result = "moderate";
            }
            else if (voc == 3)
            {
                result = "bad";
            }
            return result;
        }

        public string rStatus()
        {
            string result = "";
            if (rh >= 0 && rh < 55) // >=40
            {
                result = "good";
            }
```

```
            else if (rh >= 55 && rh < 60)
            {
                result = "moderate";
            }
            else if (rh >= 60)
            {
                result = "bad";
            }
            return result;
        }

        public string tStatus()
        {
            string result = "";
            if (t >= 20 && t < 26)
            {
                result = "good";
            }
            else if (t >= 26 && t < 30)
            {
                result = "moderate";
            }
            else if (t >= 30)
            {
                result = "bad";
            }
            return result;
        }

        public string pStatus()
        {
            string result = "";
            if (pm25 >= 0 && pm25 < 50)
            {
                result = "good";
            }
            else if (pm25 >= 50 && pm25 < 100)
            {
                result = "moderate";
            }
            else if (pm25 >= 100)
            {
                result = "bad";
            }
            return result;
        }
        public string hStatus()
        {
            return tStatus();
        }
        public string dStatus()
        {
            return rStatus();
        }

        public override string ToString()
        {
            return "\nCO2: " + c + "\nVOCs: " + v + "\nRH: " + r + "\nT: "
+ temp + "\nPM2.5: " + pm + "\nHI: " + h + "\nDP: " + d + "\nDate: "+
dateTime;
        }
```

```
        //send to db
        public string ToStringDB()
        {
            return "('" + c + "','" + v + "','" + r + "','" + temp + "','"
+ pm + "', '" + h + "','" + d + "','" + dateTime + "')";
        }
    }
}
```

A 6.4 AppAirQuality.cs

```
using System.Drawing;
using System.Drawing.Text;
using System.Windows.Forms;

namespace AirQuality
{
    class AppAirQuality
    {
       // private FrmPrincipal fp;

        //conf screen
        private float spaceInicialX;
        private float spaceInicialY;
        private float firstQueueWidth;
        private float secondQueueWidth;
        private float queueHeight;
        private float headerWidth;
        private float legendHeight;
        private float screenHeight;
        private string result;

        //labels
        private string tLabel;
        private string cLabel;
        private string dLabel;
        private string pLabel;
        private string humLabel;
        private string vLabel;
        private string hLabel;

        //sensor value labels
        private string tValue;
        private string cValue;
        private string dValue;
        private string pValue;
        private string humValue;
        private string vValue;
        private string hValue;

        //sensor units
        private string tUnit;
        private string cUnit;
        private string dUnit;
        private string pUnit;
        private string humUnit;
        private string vUnit;
        private string hUnit;

        //foot labels
        private string goodLabel;
        private string moderateLabel;
        private string badLabel;

        private string titleLabel;

        private string big;
```

```
        private string medium;
        private string small;

        private SendEmail se;

        private Button config;
        Config cf;

        SolidBrush fontBrush;
        StringFormat sf;

        public AppAirQuality(int width, int height)
        {

            tLabel = "Temperatura";
            cLabel= "CO2";
            dLabel= "Ponto de Orvalho";
            pLabel = "PM2.5";
            humLabel = "Humidade";
            vLabel = "VOCs";
            hLabel = "Sensação Térmica";

            tUnit = "°C";
            cUnit = "ppm";
            dUnit = "°C";
            pUnit ="ug/m3";
            humUnit = "%";
            vUnit = "";
            hUnit = "°C";

            goodLabel = "Bom";
            moderateLabel = "Moderado";
            badLabel = "Mau";
            titleLabel = "Monitorização da Qualidade do ar";

            big = "Big";
            medium = "Medium";
            small = "Small";

            se = new SendEmail();

            fontBrush = new SolidBrush(Color.FromArgb(8, 128, 186));
            sf = new StringFormat();

            config = new Button();
            cf = new Config();

            spaceInicialY = 0.2F * height;
            spaceInicialX = 0.1F * width;
            queueHeight = 0.35F * height;
            firstQueueWidth = 0.33F * width;
            secondQueueWidth = 0.25F * width;
            headerWidth = width;
            legendHeight = 0.10F * height;
            screenHeight = height;
            result = Resolution(width, height);
        }

        public float SpaceInicialY
        {
```

```
            get { return spaceInicialY; }
        }
        public float SpaceInicialX
        {
            get { return spaceInicialX; }
        }
        public float FirstQueueWidth
        {
            get { return firstQueueWidth; }
        }
        public float SecondQueueWidth
        {
            get { return secondQueueWidth; }
        }
        public float QueueHeight
        {
            get { return queueHeight; }
        }
        public float HeaderWidth
        {
            get { return headerWidth; }
        }
        public float LegendHeight
        {
            get { return legendHeight; }
        }
        public float ScreenHeight
        {
            get { return screenHeight; }
        }
        public string Result
        {
            get { return result; }
        }
        private string Resolution(int w, int h)
        {
            string result = "";

            if (w == 1920 && h == 1080)
            {
                result = big;
            }
            else if (w == 1024 && h == 600)
            {
                result = medium;
            }
            else if (w == 800 && h == 600)
            {
                result = small;
            }

            return result;
        }
        public void updateValues(PaintEventArgs e, Stack<SensorValues>
allValues)
        {
            Font valuesFont = new Font("Microsoft Sans Serif", 5,
FontStyle.Bold);

            if (Result.Equals(small) || Result.Equals(medium))
            {
```

```
                valuesFont = new Font("Microsoft Sans Serif", 40 * 0.5F,
FontStyle.Bold);
            }else if (Result.Equals(big))
            {
                valuesFont = new Font("Microsoft Sans Serif", 40,
FontStyle.Bold);
            }


            //Sensor value labels
            tValue = allValues.Peek().temp + tUnit;
            humValue = allValues.Peek().r + humUnit;
            hValue = allValues.Peek().h + hUnit;
            dValue = allValues.Peek().d + dUnit;
            cValue = allValues.Peek().c + cUnit;
            pValue = allValues.Peek().pm + pUnit;
            vValue = "Nível " + allValues.Peek().v + vUnit;

            //CO2 value
            SizeF size = e.Graphics.MeasureString(cValue, valuesFont);
            e.Graphics.DrawString(cValue, valuesFont, fontBrush,
SpaceInicialX + (QueueHeight * 0.9F) * 0.5F - (size.Width / 2),
(SpaceInicialY + QueueHeight) + (QueueHeight * 0.9F) * 0.40F, sf);

            //PM2.5 value
            size = e.Graphics.MeasureString(pValue, valuesFont);
            e.Graphics.DrawString(pValue, valuesFont, fontBrush,
FirstQueueWidth + SpaceInicialX + (QueueHeight * 0.9F) * 0.5F - (size.Width
/ 2), (SpaceInicialY + QueueHeight) + (QueueHeight * 0.9F) * 0.40F);

            //VOC value
            size = e.Graphics.MeasureString(vValue, valuesFont);
            e.Graphics.DrawString(vValue, valuesFont, fontBrush, 2 *
FirstQueueWidth + SpaceInicialX + (QueueHeight * 0.9F) * 0.5F - (size.Width
/ 2), (SpaceInicialY + QueueHeight) + (QueueHeight * 0.9F) * 0.40F);

            //Temperature value
            size = e.Graphics.MeasureString(tValue, valuesFont);
            e.Graphics.DrawString(tValue, valuesFont, fontBrush,
SpaceInicialX / 2 + (QueueHeight * 0.9F) * 0.5F - (size.Width / 2),
(SpaceInicialY) + (QueueHeight * 0.9F) * 0.40F);

            //Humidity value
            size = e.Graphics.MeasureString(humValue, valuesFont);
            e.Graphics.DrawString(humValue, valuesFont, fontBrush,
SecondQueueWidth + SpaceInicialX / 2 + (QueueHeight * 0.9F) * 0.5F -
(size.Width / 2), (SpaceInicialY) + (QueueHeight * 0.9F) * 0.40F);

            // Heat Index value
            size = e.Graphics.MeasureString(hValue, valuesFont);
            e.Graphics.DrawString(hValue, valuesFont, fontBrush, 2 *
SecondQueueWidth +SpaceInicialX / 2 + (QueueHeight * 0.9F) * 0.5F -
(size.Width / 2), (SpaceInicialY) + (QueueHeight * 0.9F) * 0.40F);

            //DewPoint value
            size = e.Graphics.MeasureString(dValue, valuesFont);
            e.Graphics.DrawString(dValue, valuesFont, fontBrush, 3 *
SecondQueueWidth + SpaceInicialX / 2 + (QueueHeight * 0.9F) * 0.5F -
(size.Width / 2), (SpaceInicialY) + (QueueHeight * 0.9F) * 0.40F);
```

```
        }
        public void config_click(object sender, EventArgs e) {

            cf.Show();

        }
        public void paintForm(object sender, PaintEventArgs e,
Stack<SensorValues> allValues)
        {

            e.Graphics.SmoothingMode =
System.Drawing.Drawing2D.SmoothingMode.AntiAlias;//make the graphics better
            e.Graphics.TextRenderingHint =
TextRenderingHint.AntiAlias;//text format

            Font titleFont = new Font("Microsoft Sans Serif", 5,
FontStyle.Bold);
            Font labelsFont = new Font("Microsoft Sans Serif", 5,
FontStyle.Bold);
            Font footerFont = new Font("Microsoft Sans Serif", 5,
FontStyle.Bold);

            RectangleF rCO2 = new RectangleF(spaceInicialX, spaceInicialY +
queueHeight, queueHeight * 0.90F, queueHeight * 0.90F);
            RectangleF rPM = new RectangleF(spaceInicialX +
firstQueueWidth, spaceInicialY + queueHeight, queueHeight * 0.90F,
queueHeight * 0.90F);
            RectangleF rVOC = new RectangleF(spaceInicialX + 2 *
firstQueueWidth, spaceInicialY + queueHeight, queueHeight * 0.90F,
queueHeight * 0.90F);
            RectangleF rT = new RectangleF(spaceInicialX / 2, spaceInicialY
, queueHeight * 0.90F, queueHeight * 0.90F);
            RectangleF rRH = new RectangleF(spaceInicialX / 2 +
secondQueueWidth, spaceInicialY, queueHeight * 0.90F, queueHeight * 0.90F);
            RectangleF rHI = new RectangleF(spaceInicialX / 2 + 2 *
secondQueueWidth, spaceInicialY, queueHeight * 0.90F, queueHeight * 0.90F);
            RectangleF rDP = new RectangleF(spaceInicialX / 2 + 3 *
secondQueueWidth, spaceInicialY, queueHeight * 0.90F, queueHeight * 0.90F);
// spaceInicialY + queueHeight

            //legend
            RectangleF rGood = new RectangleF(secondQueueWidth -
legendHeight * 0.2F, screenHeight * 0.9F + legendHeight * 0.50F,
legendHeight * 0.2F, legendHeight * 0.2F);
            RectangleF rModerate = new RectangleF(2 * secondQueueWidth -
legendHeight * 0.2F, screenHeight * 0.9F + legendHeight * 0.50F,
legendHeight * 0.2F, legendHeight * 0.2F);
            RectangleF rBad = new RectangleF(3 * secondQueueWidth -
legendHeight * 0.2F, screenHeight * 0.9F + legendHeight * 0.50F,
legendHeight * 0.2F, legendHeight * 0.2F);

            //colors
            Color red = Color.FromArgb(196, 2, 8);
            Color green = Color.FromArgb(0, 77, 13);
            Color yellow = Color.FromArgb(255, 228, 56);

            //Pens, brushes
            SolidBrush redBrush = new SolidBrush(red);
            SolidBrush greenBrush = new SolidBrush(green);
            SolidBrush yellowBrush = new SolidBrush(yellow);
```

```
            Pen greenPen = new Pen(green, 30);
            Pen yellowPen = new Pen(yellow, 30);
            Pen redPen = new Pen(red, 30);

            Pen greenPenLegend = new Pen(green, 5);
            Pen yellowPenLegend = new Pen(yellow, 5);
            Pen redPenLegend = new Pen(red, 5);

            string mailBody = "";

            //legend simbols
            e.Graphics.DrawEllipse(greenPenLegend, rGood);
            e.Graphics.DrawEllipse(yellowPenLegend, rModerate);
            e.Graphics.DrawEllipse(redPenLegend, rBad);

            //temperature
            if (allValues.Peek().tStatus() == "good")
            {
                e.Graphics.DrawEllipse(greenPen, rT);
            }
            else if (allValues.Peek().tStatus() == "moderate")
            {
                e.Graphics.DrawEllipse(yellowPen, rT);
            }
            else if (allValues.Peek().tStatus() == "bad")
            {
                e.Graphics.DrawEllipse(redPen, rT);
                mailBody += "\nTemperatura: " + allValues.Peek().temp +
tUnit;
            }

            //co2
            if (allValues.Peek().cStatus() == "good")
            {
                e.Graphics.DrawEllipse(greenPen, rCO2);
            }
            else if (allValues.Peek().cStatus() == "moderate")
            {
                e.Graphics.DrawEllipse(yellowPen, rCO2);
            }
            else if (allValues.Peek().cStatus() == "bad")
            {
                e.Graphics.DrawEllipse(redPen, rCO2);
                mailBody += "\nCO2: " + allValues.Peek().c + cUnit;
            }

            //Dew Point
            if (allValues.Peek().dStatus() == "good")
            {
                e.Graphics.DrawEllipse(greenPen, rDP);
            }
            else if (allValues.Peek().dStatus() == "moderate")
            {
                e.Graphics.DrawEllipse(yellowPen, rDP);
            }
            else if (allValues.Peek().dStatus() == "bad")
            {
                e.Graphics.DrawEllipse(redPen, rDP);
                mailBody += "\nPonto de orvalho: " + allValues.Peek().d +
dUnit;
```

```
            }

            //pm25
            if (allValues.Peek().pStatus() == "good")
            {
                e.Graphics.DrawEllipse(greenPen, rPM);
            }
            else if (allValues.Peek().pStatus() == "moderate")
            {
                e.Graphics.DrawEllipse(yellowPen, rPM);
            }
            else if (allValues.Peek().pStatus() == "bad")
            {
                e.Graphics.DrawEllipse(redPen, rPM);
                mailBody += "\nPM2.5: " + allValues.Peek().pm + pUnit;
            }

            //Relative Humidity
            if (allValues.Peek().rStatus() == "good")
            {
                e.Graphics.DrawEllipse(greenPen, rRH);
            }
            else if (allValues.Peek().rStatus() == "moderate")
            {
                e.Graphics.DrawEllipse(yellowPen, rRH);
            }
            else if (allValues.Peek().rStatus() == "bad")
            {
                e.Graphics.DrawEllipse(redPen, rRH);
                mailBody += "\nHumidade: " + allValues.Peek().r + humUnit;
            }

            //VOC
            if (allValues.Peek().vStatus() == "good")
            {
                e.Graphics.DrawEllipse(greenPen, rVOC);
            }
            else if (allValues.Peek().vStatus() == "moderate")
            {
                e.Graphics.DrawEllipse(yellowPen, rVOC);
            }
            else if (allValues.Peek().vStatus() == "bad")
            {
                e.Graphics.DrawEllipse(redPen, rVOC);
                mailBody += "\nVOC: " + allValues.Peek().v + vUnit;
            }

            //HI
            if (allValues.Peek().hStatus() == "good")
            {
                e.Graphics.DrawEllipse(greenPen, rHI);
            }
            else if (allValues.Peek().hStatus() == "moderate")
            {
                e.Graphics.DrawEllipse(yellowPen, rHI);
            }
            else if (allValues.Peek().hStatus() == "bad")
            {
                e.Graphics.DrawEllipse(redPen, rHI);
                mailBody += "\nSensação térmica: " + allValues.Peek().h +
hUnit;
```

```
            }

            //// Mail Alarm
            //if (mailBody.Length != 0)
            //{
            //    se.Body = "Valores muito altos\n " + mailBody + "\nData e
Hora: " + allValues.Peek().dateTime;
            //    se.SendAlarm();
            //}
            //mailBody = "";

            //labels
            if (Result.Equals(small) || Result.Equals(medium))
            {
                titleFont = new Font("Microsoft Sans Serif", 50 * 0.5F,
FontStyle.Bold);
                labelsFont = new Font("Microsoft Sans Serif", 20 * 0.5F,
FontStyle.Bold);
                footerFont = new Font("Microsoft Sans Serif", 20 * 0.5F,
FontStyle.Bold);

            }else if (Result.Equals(big))
            {
                titleFont = new Font("Microsoft Sans Serif", 50,
FontStyle.Bold);
                labelsFont = new Font("Microsoft Sans Serif", 20,
FontStyle.Bold);
                footerFont = new Font("Microsoft Sans Serif", 20,
FontStyle.Bold);
            }

            //title
            SizeF size = e.Graphics.MeasureString(titleLabel, titleFont);
            e.Graphics.DrawString(titleLabel, titleFont, fontBrush,
(headerWidth - size.Width) / 2, (screenHeight * 0.2F - size.Height) / 2,
sf);

            //CO2
            size = e.Graphics.MeasureString(cLabel, labelsFont);
//spaceInicialY + queueHeight
            e.Graphics.DrawString(cLabel, labelsFont, fontBrush,
spaceInicialX + (queueHeight * 0.9F) * 0.5F - (size.Width / 2),
spaceInicialY + queueHeight + (queueHeight * 0.9F) * 0.20F, sf);

            //PM2.5
            size = e.Graphics.MeasureString(pLabel, labelsFont);
            e.Graphics.DrawString(pLabel, labelsFont, fontBrush,
firstQueueWidth + spaceInicialX + (queueHeight * 0.9F) * 0.5F - (size.Width
/ 2), spaceInicialY + queueHeight + (queueHeight * 0.9F) * 0.20F, sf);

            //VOC
            size = e.Graphics.MeasureString(vLabel, labelsFont);
            e.Graphics.DrawString(vLabel, labelsFont, fontBrush, 2 *
firstQueueWidth + spaceInicialX + (queueHeight * 0.9F) * 0.5F - (size.Width
/ 2), spaceInicialY + queueHeight + (queueHeight * 0.9F) * 0.20F, sf);

            //Temperature
            size = e.Graphics.MeasureString(tLabel, labelsFont);
```

```
            e.Graphics.DrawString(tLabel, labelsFont, fontBrush,
spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F - (size.Width / 2),
(spaceInicialY ) + (queueHeight * 0.9F) * 0.20F, sf);

            //Humidity
            size = e.Graphics.MeasureString(humLabel, labelsFont);
            e.Graphics.DrawString(humLabel, labelsFont, fontBrush,
secondQueueWidth + spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F -
(size.Width / 2), (spaceInicialY) + (queueHeight * 0.9F) * 0.20F, sf);

            //Heat Index
            size = e.Graphics.MeasureString(hLabel, labelsFont);
            e.Graphics.DrawString(hLabel, labelsFont, fontBrush, 2 *
secondQueueWidth + spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F -
(size.Width / 2), (spaceInicialY) + (queueHeight * 0.9F) * 0.20F, sf);

            //Dew Point
            size = e.Graphics.MeasureString(dLabel, labelsFont);
            e.Graphics.DrawString(dLabel, labelsFont, fontBrush, 3 *
secondQueueWidth + spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F -
(size.Width / 2), (spaceInicialY) + (queueHeight * 0.9F) * 0.20F, sf);

            updateValues(e, allValues);

            //footer
            size = e.Graphics.MeasureString(goodLabel, footerFont);
            e.Graphics.DrawString(goodLabel, footerFont, fontBrush,
secondQueueWidth*0.02F + secondQueueWidth, screenHeight * 0.9F +
legendHeight * 0.50F + (legendHeight * 0.2F - size.Height) / 2, sf);

            size = e.Graphics.MeasureString(moderateLabel, footerFont);
            e.Graphics.DrawString(moderateLabel, footerFont, fontBrush,
2.02F * secondQueueWidth, screenHeight * 0.9F + legendHeight * 0.50F +
(legendHeight * 0.2F - size.Height) / 2, sf);

            size = e.Graphics.MeasureString(badLabel, footerFont);
            e.Graphics.DrawString(badLabel, footerFont, fontBrush, 3.02F *
secondQueueWidth, screenHeight * 0.9F + legendHeight * 0.50F +
(legendHeight * 0.2F - size.Height) / 2, sf);


            //logos
            if (Result.Equals(small) || Result.Equals(medium)) {

                Image airQuality = Image.FromFile("logoApp_.png");
                e.Graphics.DrawImage(airQuality, new PointF((headerWidth *
0.2F - airQuality.Width) / 2, (screenHeight * 0.20F - airQuality.Height) /
2));

                //PointF p = new PointF((headerWidth * 0.2F -
airQuality.Width) / 2, (screenHeight * 0.20F - airQuality.Height) / 2);
                //config.Location = Point.Round(p);
                //config.Visible = false;
                //config.Image = airQuality;
                //config.Click += config_click;
                //System.Windows.Controls.Add(config);

                Image ulsg = Image.FromFile("ulsLogo_.png");
                e.Graphics.DrawImage(ulsg, new PointF(headerWidth * 0.8F +
(headerWidth * 0.2F - airQuality.Width) / 2, (screenHeight * 0.20F -
ulsg.Height) / 2));
```

```
                Image temperature = Image.FromFile("temp_.png");
                e.Graphics.DrawImage(temperature, new PointF(spaceInicialX
+ (queueHeight * 0.9F) * 0.5F - (temperature.Width / 2), spaceInicialY +
queueHeight + (queueHeight * 0.9F) * 0.60F));

                Image co2 = Image.FromFile("co2_.png");
                e.Graphics.DrawImage(co2, new PointF(firstQueueWidth +
spaceInicialX + (queueHeight * 0.9F) * 0.5F - (co2.Width / 2),
spaceInicialY + queueHeight + (queueHeight * 0.9F) * 0.60F));

                Image dp = Image.FromFile("dp_.png");
                e.Graphics.DrawImage(dp, new PointF(2 * firstQueueWidth +
spaceInicialX + (queueHeight * 0.9F) * 0.5F - (dp.Width / 2), spaceInicialY
+ queueHeight + (queueHeight * 0.9F) * 0.60F));

                Image pm = Image.FromFile("pm_.png");
                e.Graphics.DrawImage(pm, new PointF(spaceInicialX / 2 +
(queueHeight * 0.9F) * 0.5F - (temperature.Width / 2), spaceInicialY +
(queueHeight * 0.9F) * 0.60F));

                Image humidity = Image.FromFile("hm_.png");
                e.Graphics.DrawImage(humidity, new PointF(secondQueueWidth
+ spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F - (temperature.Width /
2), spaceInicialY + (queueHeight * 0.9F) * 0.60F));

                Image voc = Image.FromFile("voc_.png");
                e.Graphics.DrawImage(voc, new PointF(2 * secondQueueWidth +
spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F - (temperature.Width / 2),
spaceInicialY + (queueHeight * 0.9F) * 0.60F));

                Image hi = Image.FromFile("hi_.png");
                e.Graphics.DrawImage(hi, new PointF(3 * secondQueueWidth +
spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F - (temperature.Width / 2),
spaceInicialY + (queueHeight * 0.9F) * 0.60F));

            }
            else if(result.Equals(big)){

                Image airQuality = Image.FromFile("logoApp.png");
                e.Graphics.DrawImage(airQuality, new PointF((headerWidth *
0.2F - airQuality.Width) / 2, (screenHeight * 0.20F - airQuality.Height) /
2));

                //PointF p = new PointF((headerWidth * 0.2F -
airQuality.Width) / 2, (screenHeight * 0.20F - airQuality.Height) / 2);
                //config.Location = Point.Round(p);
                //config.Visible = false;
                //config.Image = airQuality;
                //config.Click += config_click;
                //fp.Controls.Add(config);

                Image ulsg = Image.FromFile("ulsLogo.png");
                //e.Graphics.DrawImage(ulsg, new PointF(spaceInicialX +
airQuality.Width + headerWidth * 0.6F, spaceInicialY * 0.2F));
                e.Graphics.DrawImage(ulsg, new PointF(headerWidth * 0.8F +
(headerWidth * 0.2F - airQuality.Width) / 2, (screenHeight * 0.20F -
ulsg.Height) / 2));

                Image temperature = Image.FromFile("co2.png");
```

```
                e.Graphics.DrawImage(temperature, new PointF(spaceInicialX
+ (queueHeight * 0.9F) * 0.5F - (temperature.Width / 2), spaceInicialY +
queueHeight + (queueHeight * 0.9F) * 0.60F));

                Image co2 = Image.FromFile("pm.png");
                e.Graphics.DrawImage(co2, new PointF(firstQueueWidth +
spaceInicialX + (queueHeight * 0.9F) * 0.5F - (co2.Width / 2),
spaceInicialY + queueHeight + (queueHeight * 0.9F) * 0.60F));

                Image dp = Image.FromFile("voc.png");
                e.Graphics.DrawImage(dp, new PointF(2 * firstQueueWidth +
spaceInicialX + (queueHeight * 0.9F) * 0.5F - (dp.Width / 2), spaceInicialY
+ queueHeight + (queueHeight * 0.9F) * 0.60F));

                Image pm = Image.FromFile("temp.png");
                e.Graphics.DrawImage(pm, new PointF(spaceInicialX / 2 +
(queueHeight * 0.9F) * 0.5F - (temperature.Width / 2), spaceInicialY  +
(queueHeight * 0.9F) * 0.60F));

                Image humidity = Image.FromFile("hm.png");
                e.Graphics.DrawImage(humidity, new PointF(secondQueueWidth
+ spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F - (temperature.Width /
2), spaceInicialY + (queueHeight * 0.9F) * 0.60F));

                Image voc = Image.FromFile("hi.png");
                e.Graphics.DrawImage(voc, new PointF(2 * secondQueueWidth +
spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F - (temperature.Width / 2),
spaceInicialY + (queueHeight * 0.9F) * 0.60F));

                Image hi = Image.FromFile("dp.png");
                e.Graphics.DrawImage(hi, new PointF(3 * secondQueueWidth +
spaceInicialX / 2 + (queueHeight * 0.9F) * 0.5F - (temperature.Width / 2),
spaceInicialY + (queueHeight * 0.9F) * 0.60F));
            }

        }
    }
}
```

A 6.5 MySQLDBConnection.cs

```
using System.Data;
using MySql.Data.MySqlClient;
using System.Windows.Forms;
using System;

namespace AirQuality
{
    class MySqlDBConnection
    {
        //Atributes
        public MySqlConnection connection;
        public MySqlDataAdapter adapter;
        public string server;
        public string database;
        public string uid;
        public string password;

        //Constructor
        public MySqlDBConnection()
        {
            server = "localhost";
            database = "airqualitydb";
            uid = "root";
            password = "jasimaoIPG2018";
            string connectionString;
            connectionString = "SERVER=" + server + ";" + "DATABASE=" +
            database + ";" + "UID=" + uid + ";" + "PASSWORD=" + password +
";";
            connection = new MySqlConnection(connectionString);
        }

        //open connection to database
         public bool OpenConnection()
         {
             try
             {
                 connection.Open();
                 return true;
             }
             catch (MySqlException ex)
             {

                 switch (ex.Number)
                 {
                     case 0:
                         MessageBox.Show("Cannot connect to server.
Contact administrator");
                         break;

                     case 1045:
                         MessageBox.Show("Invalid username/password, please
try again");
                         break;
                 }
                 return false;
```

```
            }
        }

        //Close connection
        public bool CloseConnection()
        {
            try
            {
                connection.Close();
                return true;
            }
            catch (MySqlException ex)
            {
                MessageBox.Show(ex.Message);
                return false;
            }
        }

        //Insert statement
        public void Insert( string s)
        {
            string query = "INSERT INTO sensorvalue
(co2,voc,rh,t,pm25,hi,dp,date) VALUES" + s;

            //open connection
            if (OpenConnection() == true)
            {
                //create command and assign the query and connection from
the constructor
                MySqlCommand cmd = new MySqlCommand(query, connection);

                //Execute command
                cmd.ExecuteNonQuery();

                //close connection
               CloseConnection();
            }
        }
       //Select all values from the database statement
       public DataTable SelectAll()
       {

           string query = "SELECT * from sensorvalue";

           //open connection
           try
           {
               OpenConnection();
               //create command and assign the query and connection from
the constructor
               MySqlCommand cmd = new MySqlCommand(query, connection);
               //Execute command
               cmd.ExecuteNonQuery();
               adapter = new MySqlDataAdapter(cmd);
               DataTable dt = new DataTable();
               adapter.Fill(dt);
               return dt;
           }
           catch (MySqlException ex)
           {
               throw new ApplicationException(ex.ToString());
```

```
            }
            finally
            {
                //close connection
                CloseConnection();
            }

        }
        public SensorValue SelectLastEntry()
        {

            string query = "SELECT * FROM sensorvalue ORDER BY ID DESC
LIMIT 1";

            //open connection
            try
            {
                OpenConnection();
                //create command and assign the query and connection from
the constructor
                MySqlCommand cmd = new MySqlCommand(query, connection);
                //Execute command
                 cmd.ExecuteNonQuery();
                adapter = new MySqlDataAdapter(cmd);
                DataTable dataTable = new DataTable();
                adapter.Fill(dataTable);

                //SensorValues auxiliar
                SensorValue aux = new SensorValue();
                aux.c = Convert.ToInt32(dataTable.Rows[0][1]);
                aux.v = Convert.ToInt32(dataTable.Rows[0][2]);
                aux.r = Convert.ToInt32(dataTable.Rows[0][3]);
                aux.temp = Convert.ToInt32(dataTable.Rows[0][4]);
                aux.pm = Convert.ToInt32(dataTable.Rows[0][5]);
                aux.h = Convert.ToInt32(dataTable.Rows[0][6]);
                aux.d = Convert.ToInt32(dataTable.Rows[0][7]);
                //return dt.ToString();
                return aux;
            }
            catch (MySqlException ex)
            {
                throw new ApplicationException(ex.ToString());
            }
            finally
            {
                //close connection
                CloseConnection();
            }

        }

    }
}
```

A 6.6 SendEmail.cs

```
using System;
using System.Windows.Forms;
using System.Net.Mail;

/**This class use SMTP protocol to send alert email */

namespace AirQuality
{
    class SendEmail
    {
        private string serverPath;
        private string mailFrom;
        private string password;
        private string mailTo;
        private string subject;
        private string body;
        private int port;

        private MailMessage mail;
        private SmtpClient server;

        public SendEmail()
        {
            serverPath = "smtp.office365.com";
            mailFrom = "airqualityulsg@hotmail.com";
            password = "jasimaoIPG2018";
            mailTo = "jana-simao@hotmail.com";
            subject = "AirQuality alert email";
            body = "AirQuality alert email";
            port = 587;
            mail = new MailMessage();
            server = new SmtpClient(serverPath);
        }
        public SendEmail(string b)
        {
            serverPath = "smtp.office365.com";
            mailFrom = "airqualityulsg@hotmail.com";
            password = "jasimaoIPG2018";
            mailTo = "jana-simao@hotmail.com";
            subject = "AirQuality alert email";
            body = b;
            port = 587;
            mail = new MailMessage();
            server = new SmtpClient(serverPath);
        }
        public SendEmail(string s, string mf, string pwd, string mt, string
sb, string bo, int p, MailMessage mm, SmtpClient sv)
        {
            this.serverPath = s;
            this.mailFrom = mf;
            this.password = pwd;
            this.mailTo = mt;
            this.subject = sb;
            this.body = bo;
            this.port = p;
            this.mail = mm;
```

```
        this.server = sv;
    }
    public string ServerPath
    {
        get { return serverPath; }
        set { serverPath = value; }
    }
    public string MailFrom
    {
        get { return mailFrom; }
        set { mailFrom = value; }
    }
    public string Password
    {
        get { return password; }
        set { password = value; }
    }
    public string MailTo
    {
        get { return mailTo; }
        set { mailTo = value; }
    }
    public string Subject
    {
        get { return subject; }
        set { subject = value; }
    }
    public string Body
    {
        get { return body; }
        set { body = value; }
    }
    public int Port
    {
        get { return port; }
        set { port = value; }
    }
    public SmtpClient Server
    {
        get { return server; }
        set { server = value; }
    }
    public MailMessage Mail
    {
        get { return mail; }
        set { mail = value; }
    }

    public void SendAlarm()
    {
        try
        {
            MailMessage mail = new MailMessage();
            SmtpClient SmtpServer = new SmtpClient(serverPath);

            mail.From = new MailAddress(mailFrom);
            mail.To.Add(mailTo);
            mail.Subject = subject;
            mail.Body = body;
```

```
                SmtpServer.Port = port;
                SmtpServer.Credentials = new
System.Net.NetworkCredential(mailFrom, password);
                SmtpServer.EnableSsl = true;

                SmtpServer.Send(mail);
                MessageBox.Show("mail Send");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }

        }
    }
}
```

A 6.7 ThingSpeak.cs

```
using System;
using System.Collections.Generic;
using System.Net;

namespace AirQuality
{
    class ThingSpeak
    {
        private string WRITEKEY;
        private string strUpdateBase;
        private string strUpdateURI;
        WebClient webclient;

        public ThingSpeak()
        {
            WRITEKEY = "O0051IMWSCN1L8SB";
            strUpdateBase = "http://api.thingspeak.com/update";
            strUpdateURI = strUpdateBase + "?api_key=" + WRITEKEY;
            webclient = new WebClient();
        }
        public void sendToThingspeak(Stack<SensorValue> allValues)
        {
            ////send read data to thingSpeak plataform
            try
            {
            strUpdateURI += "&field1=" + allValues.Peek().c;
            strUpdateURI += "&field2=" + allValues.Peek().v;
            strUpdateURI += "&field3=" + allValues.Peek().r;
            strUpdateURI += "&field4=" + allValues.Peek().temp;
            strUpdateURI += "&field5=" + allValues.Peek().pm;
            strUpdateURI += "&field6=" + allValues.Peek().h;
            strUpdateURI += "&field7=" + allValues.Peek().d;

            webclient.UploadString(strUpdateURI, "POST", "");

            }
            catch (WebException ex)
            {
                Console.WriteLine("This program is expected to throw
WebException on successful run." +
                                    "\n\nException Message :" +
ex.Message);
                if (ex.Status == WebExceptionStatus.ProtocolError)
                {
                    Console.WriteLine("Status Code : {0}",
((HttpWebResponse)ex.Response).StatusCode);
                    Console.WriteLine("Status Description : {0}",
((HttpWebResponse)ex.Response).StatusDescription);
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```